FM-7

セブン

F-BASIC
解析マニュアル
フェーズⅡ 探究編

箕原辰夫
菊地 寿 共著
南 泰浩

SHUWA
SYSTEM
TRADING
CO.,LTD.

FM-7

F-BASIC

解析マニュアル フェーズII 探究編

SHUWA SYSTEM TRADING CO.,LTD.

まえがき

ＦＭ－７は，１９８２年後半にＦＭ－８の姉妹機として発売され，以来人気を呼んで現在パソコンにおけるベストセラーの地位を保っております．ＦＭ－７のＣＰＵ「６８０９」は「究極の８ビットＣＰＵ」と呼ばれ，その性能が高く評価されている反面，未だにユーザに馴染みの薄いＣＰＵであるため，システム内部の解析もあまりなされていないようです．特に，システムの主要部分を占めるインタプリタの内容がユーザに解放されていないのは残念でなりません．

本書は，この「ＢＡＳＩＣインタプリタ」という巨大な思想体系に対する１つの解釈を与えるものです．どんな思想体系にも様々なアプローチがあり，複数の解釈が存在するように，我々の解釈とは異なった角度からインタプリタの解析がなされても然るべきことと思いますが，本書を起点としてユーザが自分自身の解釈を打ち立てようとする探究心を高揚させていって下されば，それで我々は満足です．

このような観点から，本書では，今までどの本も取り扱わなかったＢＡＳＩＣインタプリタの根幹を探究することに焦点を絞り，基本的なルーチンの解析から始まって，数値演算のアルゴリズムにまで手を伸ばすことにしました．特に，根幹に触れる部分については，初心者の方々でもわかるよう解説に工夫を施しました．

これが契機となって，多くのユーザが，閉じ込められた空間にある各ルーチンを解放し，ＲＯＭ内インタプリタという限定された空間に託された思想を浮き彫りにしていく指南の役目を果たせれば，これ以上にない光栄と思う次第です．

本書の執筆・出版に当たって，お世話になりました水上英樹氏，渡部孝子氏，池田正昭氏，宮田僚司氏，山内　直氏に深く感謝いたします．

１９８３年　１２月

箕原辰夫
菊地　寿
南　泰浩

目　次

第0章　本書を読むに当たって…… 1

（1）メモリの表記法について…… 3
（2）FAC（フローティングアキュムレータ）について…… 4
（3）SD（ストリングディスクリプタ），SSP（ストリングスタックポインタ）について…… 6
（4）SAC（ストリングアキュームレータ）…… 6
（5）FD（ファイルディスクリプタ）とSFD（システムファイルディスクリプタ）について…… 7
（6）汎用読込みルーチン　＄D2および＄D8について…… 7
（7）その他の主なサブルーチンについて…… 8

第1章　BASICの起動…… 9

1－1　BOOT・ROMからのBASICの起動…… 11
1－1－1　BOOT・ROMの構成…… 11
1－1－2　BASIC起動ルーチン…… 11
1－1－3　BOOT・ROM内のサブルーチン使用法…… 15
1－2　IPL（Initial Program Loader）…… 15
1－3　BASIC・COLD・STARTルーチン…… 18
1－3－1　主な動作…… 18
1－3－2　ワークエリアの設定…… 19
1－3－3　サブルーチン概説…… 23
1－4　DISK・BASIC・INITIALIZE…… 27
1－4－1　主な動作…… 27
1－4－2　ユーザ・プログラムのオート・スタート…… 27
1－4－3　ドライブ数，ファイルバッファ数の決定…… 31
1－4－4　DISKモードでのワークエリアの設定…… 32

1—4—5　ワークエリア設定後の動作……………………………36
1—5　HOT・START……………………………………………36

第2章　BASICインタプリタの骨格……………………37

はじめに……………………………………………………………39
2—1　BASICプログラムの格納形式…………………………40
2—2　変数の格納形式（単純変数と配列変数）………………45
2—2—1　単純変数の格納形式…………………………………46
2—2—2　配列変数の格納形式…………………………………56
2—3　BASICメインルーチン…………………………………60
2—3—1　BASICメインルーチンの構造………………………60
2—3—2　メインルーチンにおける重要なワークエリアなど……62
2—3—3　メインルーチン・プロンプト表示部…………………64
2—3—4　メインルーチン・テキスト作成部
（テキスト作成ルーチン）…………66
（1）　AUTO編集中の処理ルーチン…………………………70
（2）　行番号読込みルーチン　$9162………………………73
（3）　プロテクトチェックルーチン…………………………75
（4）　1行翻訳ルーチン　——中間言語作成工場——　……76
（5）　行番号サーチルーチン…………………………………99
（6）　リンクポインタ修正ルーチン…………………………100
2—3—5　メインルーチン・解読実行部（解読実行ルーチン）‥101
2—3—6　Breakチェックルーチン……………………………107
2—3—7　音関係の初期化ルーチン……………………………107

第3章　システム的コマンドと
そのサブルーチン…………………………109

3—1　NEWコマンド処理ルーチン
——テキスト消去・変数消去・
各種の初期設定を行うルーチン——　……111
3—2　RESTORE処理ルーチン………………………………114
3—3　RUN・GOTO・GOSUB処理系ブロック………………115

3—4　END・STOP・Break処理系ブロック……118
3—5　CONTコマンド処理系……119
3—6　RETURN……120
3—6—1　スタック上のネスティング情報を検索するルーチン…121
3—7　DATA・REM・ELSE処理と非実行文のスキップ……122
3—8　LET文（代入文）処理系
——変数サーチ，式の評価——……123
3—8—1　変数サーチ
——変数の検索・登録などを行うルーチン——…126
3—8—2　式の評価ルーチン……129
3—8—3　単項式の評価ルーチン……136

第4章　入出力主要部……141

4—1　入出力とファイル……143
4—1—1　ファイルとシステムファイルディスクリプタ（SFD）……143
4—1—2　入出力のバッファ……146
4—1—3　ドライブテーブル（DRVTBL）……148
4—1—4　ファイルバッファ（FBUF）……150
4—1—5　入出力のジャンプテーブル……152
4—2　入出力応用ルーチンの概要……155
4—2—1　セーブ（SAVE）ルーチンの概要……155
4—2—2　ロード（LOAD）ルーチンの概要……156
4—2—3　オープン（OPEN）ルーチンの概要……157
4—2—4　クローズ（CLOSE）ルーチンの概要……158
4—3　基本的入出力ルーチン……160
4—3—1　ジャンプテーブルを使う基本的なルーチン……160
4—3—2　ファイル関係ルーチン……163
4—3—3　基本的入出力ルーチンを利用したデータ入出力ルーチン……165
4—3—4　FIRQとAbortルーチン……170
4—3—5　サンプル……171
4—4　カセット入出力ルーチン……174

4－4－1　カセット入出力ルーチンの概要とワークエリア………174
4－4－2　カセット　オープン（ＯＰＥＮ）ルーチン……………176
4－4－3　カセット　クローズ（ＣＬＯＳＥ）ルーチン…………177
4－4－4　カセット出力ルーチン……………………………………178
4－4－5　カセット入力ルーチン……………………………………181
4－4－6　その他の主なカセット関係ルーチンの概要……………184
4－4－7　サンプル……………………………………………………185
4－5　ディスク入出力ルーチン……………………………………………187
4－5－1　ディスク入出力ルーチンの概要とワークエリア………187
4－5－2　ディスク　オープン（ＯＰＥＮ）ルーチン……………188
4－5－3　ディスク　クローズ（ＣＬＯＳＥ）ルーチン…………190
4－5－4　ディスク入出力ルーチン…………………………………191
4－5－5　その他の主なディスク関係ルーチンの概要……………194
4－5－6　サンプル（ディスクエディタ）…………………………199
4－6　コンソール入出力・プリンタ出力ルーチン……………………203
4－6－1　ＢＩＯＳ関連ルーチン……………………………………203
4－6－2　コンソール１バイト入出力ルーチン……………………205
4－6－3　その他の主なコンソール入出力関係ルーチンの概要………207
4－6－4　プリンタ出力ルーチン……………………………………211

第５章　数値演算……………………………………………………215

5－1　マシン語上での整数の表現…………………………………………217
5－2　演算の方法……………………………………………………………219
5－2－1　数値型判断ルーチン………………………………………220
5－2－2　符号判定ルーチン…………………………………………220
5－2－3　正規化ルーチン……………………………………………221
5－2－4　切捨てルーチン……………………………………………223
5－2－5　数値型変換ルーチン………………………………………224
5－2－6　数値を転送するルーチン…………………………………228
5－2－7　数値→文字列変換ルーチン………………………………232
5－2－8　文字列→数値変換ルーチン………………………………233
5－3　２項演算………………………………………………………………234
5－3－1　実数の加算を行うルーチン………………………………234

5―3―2　実数の減算を行うルーチン……239
5―3―3　実数の乗算を行うルーチン……240
5―3―4　実数の除算を行うルーチン……243
5―3―5　整数の加算・減算を行うルーチン……247
5―3―6　整数の乗算を行うルーチン……248
5―3―7　整数の除算・余りを求めるルーチン……250
5―4　単項演算……253
5―4―1　多項式演算を行うルーチン……255
5―4―2　ＳＩＮ・ＣＯＳを求めるルーチン……259
5―4―3　ＴＡＮを求めるルーチン……262
5―4―4　ＡＴＮを求めるルーチン……263
5―4―5　ＥＸＰを求めるルーチン……265
5―4―6　ＬＯＧを求めるルーチン……267
5―4―7　ＳＱＲ・べき乗を求めるルーチン……269
5―4―8　ＩＮＴを求めるルーチン……272
5―4―9　ＦＩＸを求めるルーチン……275
5―4―10　ＡＢＳを求めるルーチン……276
5―4―11　ＳＧＮを求めるルーチン……277

第6章　付　録……**279**

6―1　ワークエリアー覧表……281
6―2　メモリマップ……292
6―3　ＤＩＳＫ・ＢＡＳＩＣのメモリマップ……303
6―4　中間コードー覧表（コード順）……304
6―5　実装図・回路図……308

第０章　本書を読むに当たって

本書を読むに当たって最低限必要なことは，６８０９ＣＰＵの基本的命令セットとそのアセンブリ言語による表現形式についての知識です．特に，豊富なアドレッシングモードを持っている点と，フラグが頻繁に変化する点で，Ｚ８０など他の８ビットＣＰＵとは大幅に違っていますので注意を要します．アドレッシングモードの中でも，とりわけ，**間接アドレッシング**および**プログラムカウンタ相対アドレッシング**（後者はインタプリタ内では使用されていませんが，位置自由〔ポジションインデペンダント〕なプログラムを作るためには必要不可決なもので，サンプルでは多用しました）の２つのモードに関しては理解を深めておく必要があります．

本文および図中で用いた語句，特に略語については，ほとんど一般に通用しているもの（例えば，Ａレジスタを Areg，AccA と略するなど）を使用しましたので，特に説明は行いません．ただ，ＦＡＣ・ＳＤ・ＳＳＰ・ＳＡＣ・ＦＤ・ＳＦＤなどの用語およびインタプリタ側から参照されるメモリの表記法については，本書特有の定義に従っていますので以下に説明を加えていきます．また，文中の「コマンド」と「ステートメント」は，本来，厳密に区別すべきものかもしれませんが，両者とも「文」と同義で使用している場合もあります．なお，一部掲載したＲＯＭ内ソースリストについては，わかりやすくするために完全なアセンブリ表記形式になっていない部分があります．その他の事柄については「Ｆ－ＢＡＳＩＣ文法書」，「ＦＭ－７　ユーザーズマニュアル　システム仕様」，「同　システム解説」，および「Ｆ－ＢＡＳＩＣ解析マニュアル　フェーズＩ――基礎編――」等を適宜御参照下さい．

（１）　メモリの表記法について

メモリの内容については，値そのものと区別するためにカッコでくくります．例えば，次のようになります．

- （０５ＡＣ）は，＄０５ＡＣ番地に格納されている１バイトデータ，またはその領域のことを指します．
- （０３３Ａ：０３３Ｂ）は，＄０３３Ａの内容を上位バイト・＄０３３Ｂの内容を下位バイトとする２バイトのデータ，またはその領域のことを指します．
- （０４３Ｄ～０５３Ｃ）は，＄０４３Ｄ～＄０５３Ｃに入る一連のデータ列，またはその領域のことを指します．

ただし，インタプリタ内では常に，ＤＰ（ダイレクトページレジスト）は＄００に固定されています（ＢＩＯＳ内では，＄ＦＤに固定）から，ダイレクトアド

レッシングにおける上位バイトは必ず＄００となるため，＄００００～００FF
の範囲のワークエリアについては，例えば，（００BF）→（BF），（００３３：００３４）→（３３：３４），（００７４～００７C）→（７４～７C）などというように上位バイトを省いた略記法を多く用いました．また，このエリアのことをページゼロとも呼びます．

（2）　FAC（フローティングアキュームレータ）について

インタプリタ内では，整数型・単精度型……などの種々の演算操作を効率良く実行するため，**フローティングアキュームレータ**（Floating Point Accumulator：浮動小数点演算用アキュームレータ）と呼ぶ8バイトの仮想レジスタをワークエリア上に設けており，これを**FAC**と名付けます．FACは3つあります．

- **FAC1（74～7C）**　演算の主軸となるレジスタです．演算結果は全てここに生成されます．
- **FAC2（82～8A）**
  ２項演算でFAC１と共に用いられます．
- **FAC3（28～30）**
  実数の乗算と除算のみに使用され，仮数部だけが用いられます（指数部・符号部のエリアも確保されていますので，ユーザによる使用は可能です）．

FAC1および**FAC2**については，CPU内のレジスタと同様の命令を行うサブルーチンが設けられており，「PSHS　FAC１」(＄９３E８)，「PULS　FAC２」(＄９４１１)，「LDFAC１　,X」(＄B３０１)，「STFAC１　,X」(＄B３３７)，「LDFAC２　,X」(＄B１BA)，「TFR　FAC２　,FAC１」(＄B３５F)，「TFR　FAC１　,FAC２」（＄B３８０)，「EXG　FAC１　,FAC２」(＄９３６４)，「CMPFAC１　,X（実数のみ）」(＄B３EC）が用意されています（仮想レジスタと呼ばれる意味がこのようなことからも理解できます）．

なお，**＄００１７**番地にはFAC１の**数値型**を示す**識別子**（整数型＄０２，単精度型＄０４，倍精度型＄０８，文字型＄０３）が，**＄００５D**にはFAC２の数値型を示す識別子がそれぞれ格納されるレジスタとして用いられています．

また，FACに関連して，＄００８Bは演算の**符号判定用レジスタ**（５－３節）として，＄００８CはFACの**保護バイト**(５－２－３参照)として用いられ

ていますので注意して下さい．

次に，ＦＡＣ１を例にして，エリアの様子を見ることにします．

●**整数型の場合**は，１６ビットの符号付き２進数として表されます．従って＄０００１が**1**に，＄ＦＦＦＦが**－1**に対応し，＄７ＦＦＦが**３２７６７**に，＄８０００が**－３２７６８**に対応します．

●**単精度型の場合**は次のように各部が分かれます．

指数部　実数をＸ＝±０．△△△△……△×2^nと表したときのnが**＄８０**だけゲタを履かされて入ります．すなわち＄０１～＄８０が2^{-127}～2^0に対応し，＄８０～＄ＦＦが2^0～2^{+127}に対応します．＄００は０に対応します．

仮数部　上記△△△△…△（△は１か０）の部分が順に**＄００７５**番地の先頭ビットから代入されます（ただし，最上位ビットが常に**1**になるように入ります．正規化５－２－３参照）．

符号部　**＄００７Ｃ番地の最上位ビット**が符号を表します（正：０，負：１）．

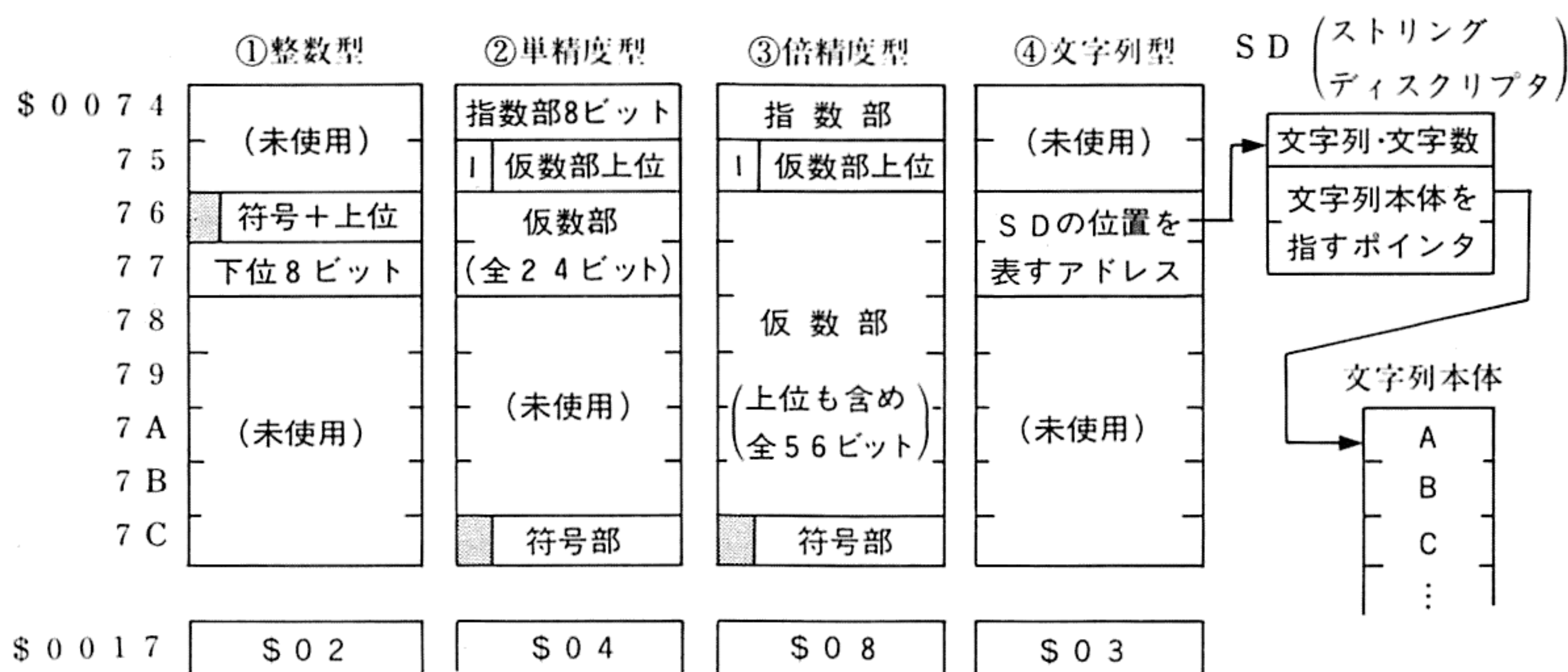
●**倍精度型の場合**は単精度型の場合とほとんど同様ですが，型を示す＄００１７番地に＄０８が入ります．さらに，単精度型では使用されていなかった下位４バイトを仮数部として使用します．

●**文字列型の場合**はＳＤ（ストリングディスクリプタ）の位置を指し示す**ポインタ**が入ります．主としてＳＡＣ内のＳＤを指すことになります．

図(1)－１．ＦＡＣ１（００７４～００７Ｃ）の構造

	①整数型	②単精度型	③倍精度型	④文字列型
＄００７４	（未使用）	指数部8ビット	指 数 部	（未使用）
７５		1 仮数部上位	1 仮数部上位	
７６	符号＋上位	仮数部（全２４ビット）	仮 数 部（上位も含め全５６ビット）	ＳＤの位置を表すアドレス
７７	下位８ビット			
７８	（未使用）	（未使用）		（未使用）
７９				
７Ａ				
７Ｂ				
７Ｃ		符号部	符号部	
＄００１７	＄０２	＄０４	＄０８	＄０３

ＳＤ（ストリングディスクリプタ）：文字列・文字数／文字列本体を指すポインタ

文字列本体：Ａ　Ｂ　Ｃ　⋮

※図中，▒は符号を表すビットで，「０:正；１:負」を表す．

（3）　ＳＤ（ストリングディスクリプタ），ＳＳＰ（ストリングスタックポインタ）について

文字列型データは不定長であるため，**間接表現**が採用されます．この間接表現を担っているのがストリングディスクリプタ（String Descriptor：ＳＤと略す）と名付けた3バイトのデータブロックで，最初の１バイトには文字列の**文字数**（データ長）が入り，残りの2バイトは実際の文字列が格納されている位置を示す**ポインタ**となります．インタプリタでは，実際の文字列を，主として文字列スタックエリアと呼ぶ領域（この領域の大きさはＣＬＥＡＲ文の第１パラメータで，また領域の先頭（下限）はＣＬＥＡＲ文の第2パラメータマイナス１で与えられます）で扱います．このとき，使用済領域と未使用領域との境界を指すポインタが必要であり，ワークレジスタ（４１：４２）がこれに相当します．これは，文字列スタックエリア（String Stack Area）のポインタですから，ストリングスタックポインタ(String Stack Pointer：ＳＳＰと略す)と命名いたしました．これについては，２－２－１節で詳細を御覧下さい．

（4）　ＳＡＣ（ストリングアキュームレータ）

ストリングアキュームレータ（String Accumulator：ＳＡＣと略す）という名称は，ＦＡＣをもとにして名付けたもので，本書の中だけで使用されている特有な名称です．前述のように，文字列は**ＳＤ→文字列スタックエリア**の２段構えで表現されますが，文字列関数や文字列演算（加算式）の処理を行う場合には，作業領域が必要不可欠になります．実際の文字列本体は文字列スタックエリアに置くこととして，その本体を表現するＳＤを置く場所が問題となります．ＦＭ－７では，このような作業用のＳＤを置く場所として（０５７８～０５９５）の３０バイト（ＳＤ１０個分）が用意されており，本書ではこの１０個分のエリアに対して**ＳＡＣ１，ＳＡＣ２，……，ＳＡＣ１０**という名称を与えます．

現在使用しているＳＡＣの番地，および次に使用すべきＳＡＣの番地を指すポインタが（２１：２２）および（１Ｅ：１Ｆ）であり，作業領域が増えるたびにペアでインクリメント（＋３），１つの作業が終わるたびにデクリメント（－３）されます．ＳＡＣポインタのインクリメント・デクリメントは，ＦＡＣのプッシュ・プルに相当すると考えて下さい．

また，ＳＡＣは１０個しかないので，例えば，

Ａ＄＝"　"＋（"　"＋（"　"＋……（"　"＋"ＡＢＣ"））……））

（9重のカッコ）

などと行うと「String Formula Too Complex」エラーが発生します．

（５） ＦＤ（ファイルディスクリプタ）とＳＦＤ（システムファイルディスクリプタ）について

ファイルディスクリプタ（File Descriptor：ＦＤと略す）の意味は，「Ｆ－ＢＡＳＩＣ文法書２．８」のファイルディスクリプタのことを指しています．しかし，システムファイルディスクリプタ（System File Descriptor：ＳＦＤと略す）はインタプリタ内部のもので，Ｆ－ＢＡＳＩＣプログラム上のものとは異なりますので，本書では「システムー」を接頭語として付けて区別しました．ＳＦＤについては４－１－１節に詳細な解説を行っています．

（６） 汎用読込みルーチン　＄Ｄ２および＄Ｄ８について

ワークエリアの（Ｄ２～ＤＤ）には，テキストをはじめ，バッファ等に蓄積された様々なデータの読込みの心臓部をなす非常に重要なサブルーチンが存在します（エントリは＄Ｄ２および＄Ｄ８の２種類があります）．このルーチンは，ＢＡＳＩＣ起動時に上記アドレスに設定され，インタプリタ内のほとんどの処理系から呼び出されているため，これを破壊するとインタプリタは全て動かなくなります．このルーチンの働きは，ルーチン内部に含んでいるポインタ（Ｄ９：ＤＡ）が指し示しているアドレスの内容をＡレジスタに読み込んで，そのデータの種類に応じてフラグを立てることです．＄Ｄ２と＄Ｄ８の違いは，読込みに先立って，ポインタ（Ｄ９：ＤＡ）をインクリメントする（＝＄Ｄ２）か，しない（＝＄Ｄ８）かの違いだけです．読込みを終えた後，＄Ｃ７６０に飛んで，

① Ａレジスタ内のデータが数字を表す文字コード（＄３０～＄３９）であればキャリーフラグをセットします〔数字の検出〕．
② Ａレジスタ内のデータがＮＵＬＬまたはコロンを表す文字コード（＄００または＄３Ａ）であればゼロフラグをセットします〔区切文字の検出〕．
③ Ａレジスタ内のデータが空白のコード（＄２０）である場合には再び＄Ｄ２に戻って次の番地からデータを読み込み，空白以外の文字が出てくるまでポインタ（Ｄ９：ＤＡ）を更新しつつ読み飛ばします〔空白▼␣▼のスキップ〕．

などの後処理を同時に行います．この，＄００Ｄ２から始まって＄Ｃ７６０を経て＄Ｃ７６Ｅに終わる，**汎用読込みルーチン**は，実に巧妙にできており，わずか２７バイトで上記のすべての機能を備えているだけでなく，結果が格納されるＡおよびＣＣレジスタを除いた全てのレジスタがそのまま保存されます．特にフラグの立て方や，＄Ｄ２～Ｄ７におけるプレインクリメントの行い方などは技巧

的で効率良くできています（この部分はユーザにとって大いに参考になる点ですので，ソースリストを載せておきます．キャリーフラグの立て方に注目して下さい）．また，ページゼロのエリア上に書かれていることにより，ＤＰ＝＄００である限り「ＪＳＲ　＜＄Ｄ８」，「ＪＭＰ　＜＄Ｄ２」等というように2バイト命令で呼び出すことができるのも大きな特色です．

ソースリスト　（▼＜▼は８ビットオペランドを強制する記号です）

```
$00D2  INC  <$DA              $C760  CMPA  #$3A ▼:▼
  D4   BNE  <$D8                 62  BHS   $C76E
  D6   INC  <$D9                 64  CMPA  #$20
$00D8  LDA (エクステンド)          66  BNE   $C76A
  D9   ┌汎用読込み ┐             68  JMP   <$D2
  DA   └    ポインタ┘          $C76A  SUBA  #$30
  DB   JMP  $C760                6C  SUBA  #$D0
                              $C76E  RTS
```

（７）　その他の主なサブルーチンについて

ワークエリア（ＤＥ～Ｅ０）には，ＢＩＯＳへのＪＭＰ命令（ＪＭＰ　＄Ｆ１７Ｄ）が書かれていますので，インタプリタ内からのＢＩＯＳコールはほとんど「ＪＳＲ　＜＄ＤＥ」で行われます．

＄９２８Ｃから始まる一連の**文字コードチェック**ルーチン（左カッコ：９２８Ｆ，右カッコ：９２８Ｃ，コンマ：９２９２，Breg内のコード：９２９４）については３－８－３節で詳細に述べますので，ここでは省略します．

なお，**アルファベットチェック**（＄９４ＦＤ），**数字チェック**（＄９５０６）および**小文字⇨大文字変換付きアルファベットテスト**（＄Ｃ４９０）の各サブルーチンについては，２－３－４節（４）の〔14〕で解説します．

担当：　　箕原辰夫……第１章，第４章
　　　　　菊地　寿……第２章，第３章
　　　　　南　泰浩……第５章

第１章　BASICの起動

1—1　BOOT・ROMからのBASICの起動

６８０９マイクロプロセッサでは，リセット信号がかかった後，リセットベクトルとして（FFFE：FFFF）より，実行開始アドレスを取り込みます．FM－７では，ここに＄FE００が書かれているため，＄FE００が，実行開始アドレスとなる訳ですが，＄FE００からは，BOOT・ROMの領域が入っています．BOOT・ROMは，DIPスイッチにより切り換えられ，DIPスイッチの１，２番をON状態にするとBASICモードになり，BOOT・ROMもBASIC用になります．

1—1—1　BOOT・ROMの構成

BOOT・ROMは大きく分けると，BASIC起動ルーチン，RESTORE（シークトラック０）ルーチン，DWRITE（フロッピ１セクタライト）ルーチン，DREAD（フロッピ１セクタリード）ルーチンに分かれています．また，スタック領域として＄FC７F以下のRAM領域を利用しています（図1・1・1参照）．

ディスク用のルーチンは，そのエントリ番地を直接呼ぶこともできますが，図1・1・1で示したように，RESTOREは＄FE０２，DWRITEは＄FE０５，DREADは＄FE０８といったエントリが用意されていますので，これらを使用するようにして下さい．

1—1—2　BASIC起動ルーチン

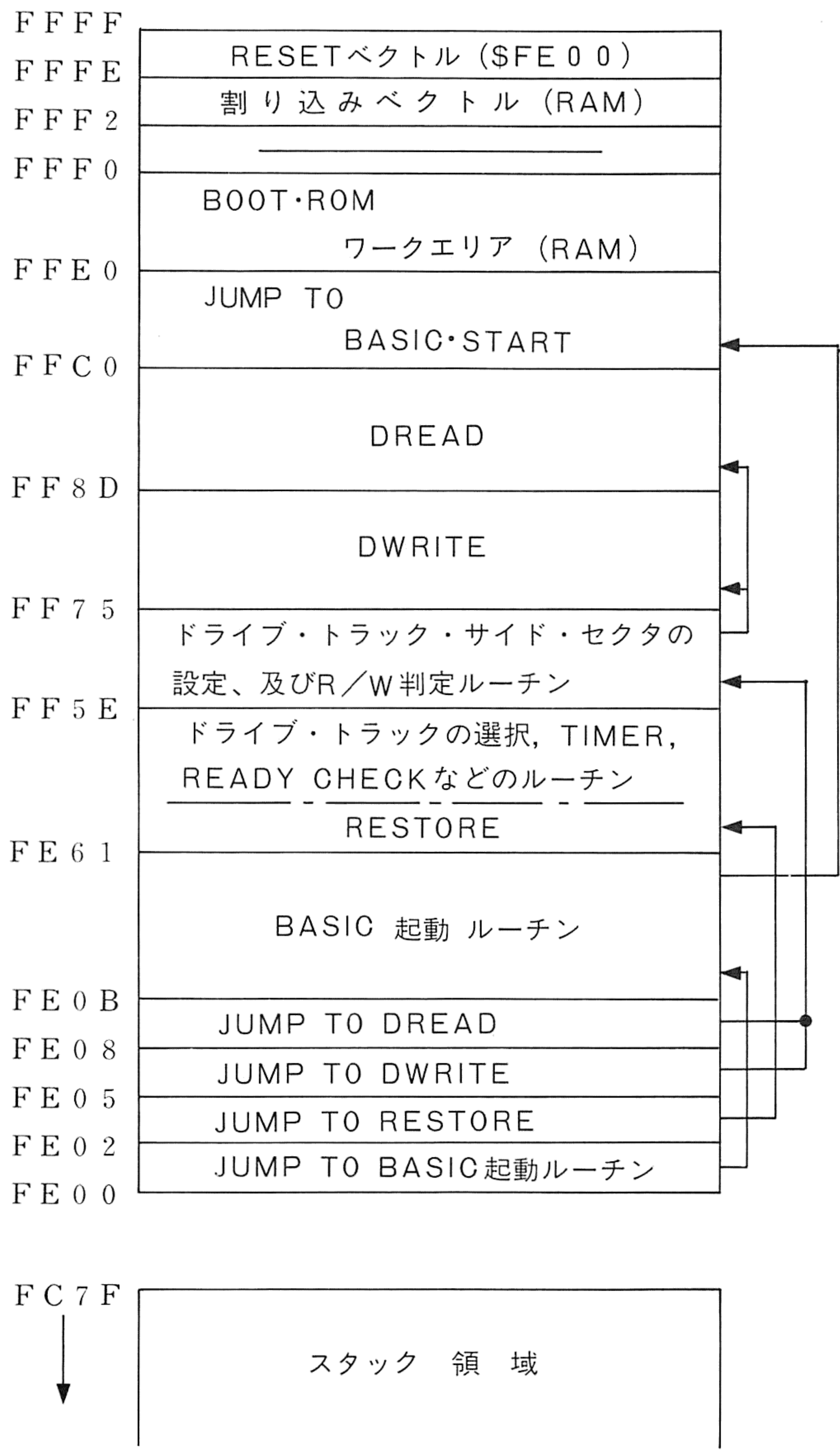
＄FE００番地のブランチ命令により，実際に起動ルーチンが始まるのは，＄FE０B番地からです．また，図1・1・1にも示したように，一連の動作が終わってからは＄FFC０番地に飛び，そこで特定のI／Oポートの設定をした後に，BASIC（または，IPL）へとジャンプしていきます．

＄FFC０番地に飛ぶまでの起動ルーチンの主な動作を以下に示します．

(１)スタックポインタとして＄FC７Fを設定します．

(２)I／Oポート（＄FD０４）のビット１を参照し，ブレーク・キーが押されていれば，Xレジスタに BASICのホットスタートのエントリ番地を代入します．また，押されていなければ次の動作を行います．

図1・1・1　BOOT・ROMの構成

アドレス	内容
FFFF	RESETベクトル（$FE00）
FFFE	割り込みベクトル（RAM）
FFF2	――――
FFF0	BOOT・ROM ワークエリア（RAM）
FFE0	JUMP TO BASIC・START
FFC0	DREAD
FF8D	DWRITE
FF75	ドライブ・トラック・サイド・セクタの設定、及びR／W判定ルーチン
FF5E	ドライブ・トラックの選択，TIMER，READY CHECKなどのルーチン / RESTORE
FE61	BASIC 起動 ルーチン
FE0B	JUMP TO DREAD
FE08	JUMP TO DWRITE
FE05	JUMP TO RESTORE
FE02	JUMP TO BASIC起動ルーチン
FE00	
FC7F ↓	スタック 領 域

(i)I／Oポート（$FD05）のビット0を参照し，拡張モジュールがあればDREAD（＝$FE08）をコールし，エラーがない限り，IPL (Initial Program Loader)をディスクから$0100番地以降にロードし，Xレジスタにエントリ番地を代入します．

(ii)拡張モジュールのない場合は，XレジスタにBASICのコールドスタートのエントリ番地を代入します．

Xレジスタに代入されるエントリ番地と，それが書かれているROMのアドレスは次の通りです．

ホットスタート	：	$8684	(FBFC：FBFD)
コールドスタート	：	$848B	(FBFE：FBFF)
IPL	：	$0100	

IPLをディスクからロードする際にエラーがあったとき，そのエラーがDisk Not Readyの場合は，拡張モジュールがないときと同じ動作をします．しかし，それ以外のエラーの場合はRESTORE(＝$FE02)をコールして，ドライブヘッドをトラック0の位置にセットをした後，IPLを再度ディスクから読み込もうとします．ただし，この動作を計5回繰り返した後もエラーが起こるようであれば，拡張モジュールがない場合と同じ動作をします．

このDREADやRESTOREなどを呼ぶ際には，BIOSと同様にRCB(Request Control Brock)が必要で，RCBのデータがDREAD用には$FE51～$FE58番地に，またRESTORE用には$FE59～$FE60番地に格納されています．

$FFC0番地からは，次の動作をします．

(1)DP (Direct Page) レジスタに，$00をセットする．

(2)(FD07) つまり，RS－232Cインタフェースのコマンドレジスタにリセット命令を送る．

(3)(FD25)，(FD27)，(FD29)，(FD2B)にもリセット命令を送る．

(4)Xレジスタの指すアドレス（各々のエントリ番地）へジャンプする．

なお（FD07）は，本体内のRS－232Cインタフェースのコマンドレジスタですが，(FD25）からの4つは拡張ユニット内のRS－232Cインタフェースのコマンドレジスタを示します．

図1・1・2　BASIC起動ルーチン

$FE0B

START

DP←$FD

SP←$FC7F

Break Key?

YES

NO

EXTDET?

無

有

DREAD

ERROR?

無

有

Drive Not Ready エラー?

YES

NO

5回目?

YES

NO

RESTORE

$FFC0

DP←$00

(FD07)←RESET
(FD25)←RESET
(FD27)←RESET
(FD29)←RESET
(FD2B)←RESET

JUMP (X)

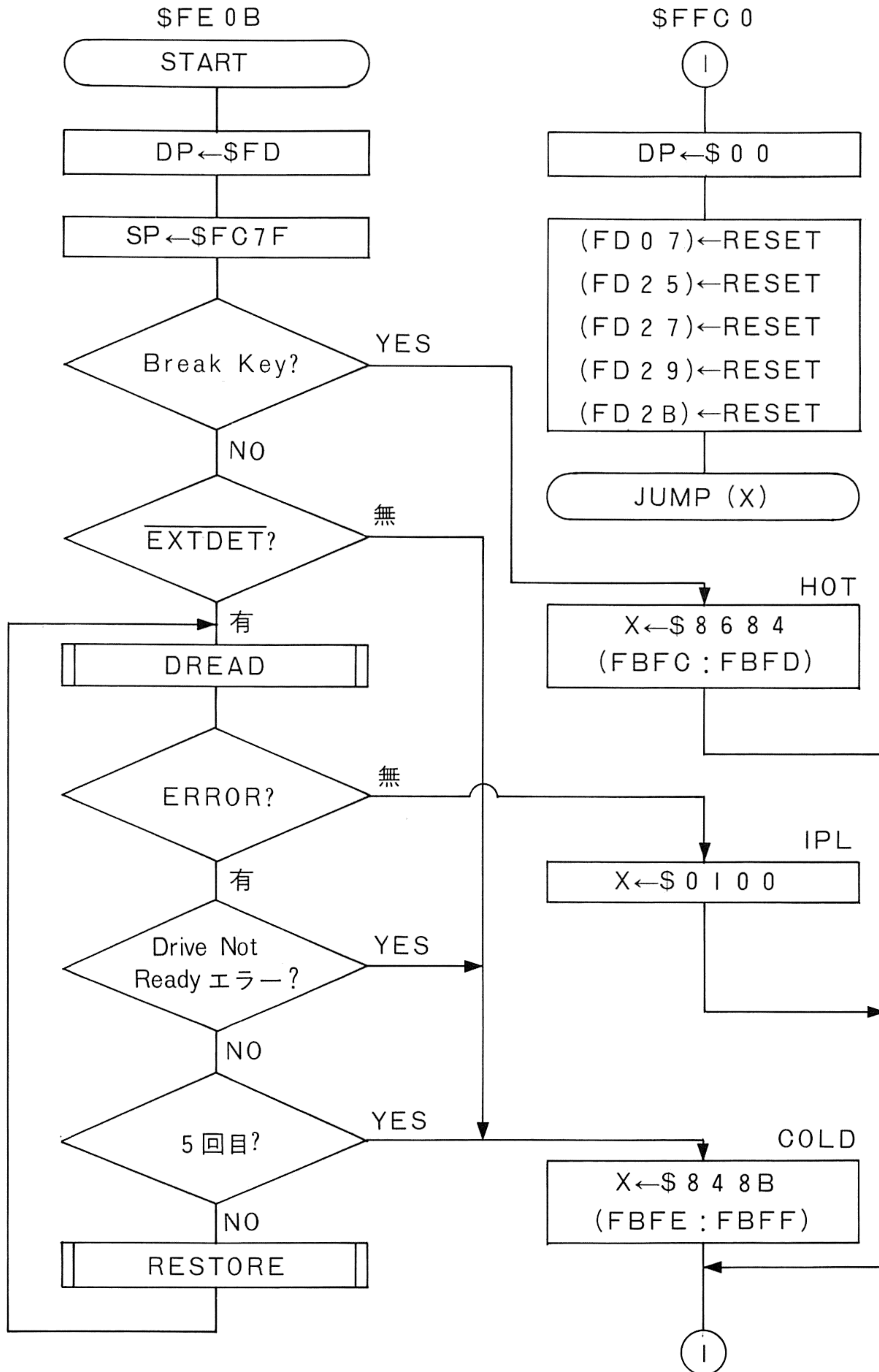
HOT

X←$8684
(FBFC : FBFD)

IPL

X←$0100

COLD

X←$848B
(FBFE : FBFF)

※$\overline{\text{EXTDET}}$は拡張モジュール無／有フラグ(FD05のビット0)

1－1－3　ＢＯＯＴ・ＲＯＭ内のサブルーチン使用法

ＢＯＯＴ・ＲＯＭ内のディスク関係のサブルーチンは，ＢＩＯＳからも呼ばれており，その中身は同一のものです．従って，ＢＩＯＳのサブルーチンと使い方はほとんど同様ですが，直接このサブルーチンを使うときは次の相違点があります（ＢＩＯＳを介せば，ＢＩＯＳがこれらの処理を行うことになります）．

（１）ＤＰレジスタに＄ＦＤを代入しておく必要があること．

（２）全てのレジスタが使われ，保存されないこと．

（３）エラーステータスがセットされず，代りにＡレジスタにエラー内容が代入されて戻されること．

以上を除けば同じですので，ＲＣＢを設定し，ＸレジスタにＲＣＢの先頭アドレスを代入し各エントリに飛ばしてください（システム仕様参照）．

1－2　ＩＰＬ（Initial Program Loader）

ＤＩＳＫ・ＢＡＳＩＣでは，ＢＯＯＴ・ＲＯＭによりディスクのトラック０，セクタ１から＄０１００番地以降へと読み込まれます．その全体は，161バイトの短いプログラムのため，ＩＰＬとして２セクタ取られていますが，実際には１セクタで終わっています（セクタ２は，未使用です）．

ＩＰＬの主な動作は次の通りです．

- ＩＤセクタを読み込み，そのディスクがＦＭ－７で使えるものかどうか判断し，使えなければＲＯＭ・ＢＡＳＩＣを走らせます．
- トラック０，セクタ１５及び１６にある**ＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥ**ルーチンを＄６Ｅ００～＄６ＦＦＦに読み込みます．
- トラック０，セクタ１７～３２の**ＤＩＳＫ・ＢＡＳＩＣ**を＄７０００～＄７ＦＦＦに読み込みます．
- 何らかのエラーが起こった場合は，ＢＯＯＴ・ＲＯＭと同様に，それがDrive Not Readyエラーかどうか調べ，そうである場合はＲＯＭ・ＢＡＳＩＣを起動させます．また，それ以外のエラーのときは計５回読み取りを繰り返しますが，それでもエラーの起こる場合はＲＯＭ・ＢＡＳＩＣを起動させます．

ＩＰＬは，ＩＤセクタやＤＩＳＫ・ＢＡＳＩＣを読み込む際，ＢＯＯＴ・ＲＯＭの中のディスク用ルーチンを使っていますのでＲＣＢが必要となります．

図1・2・1　IPL（$0100～$01A0）

$0100

IPL

割込み　マスク
S←　$0100
DP←　$FD

IDセクタの読込みおよびチェック

DBA＝$01A1
SCT＝3

X ← $0112

DISK READ

ID＝"S"　NO
YES

DISK・BASIC・INITIALIZEルーチンの読込み

DBA＝$6E00
SCT＝15

X ←$0102

DISK READ

2回目 ?　YES
NO

DBA←DBA＋$0100
SCT←SCT＋1

DISK・BASICの読込み

DBA＝$7000
SCT＝17

X ← $010A

DISK READ

16回目 ?　YES
NO

DBA←DBA＋$0100
SCT←SCT＋1

$0178

DISK READ

RESTORE
(BOOT・ROM)

Drive Not Ready エラー?　YES
NO

DREAD
(BOOT・ROM)

Error ?　無
有

Drive Not Ready エラー?　YES
NO

5回目 ?　NO　YES

RETURN

ROM・BASIC

A ← 0
DP ← 0

DISK・BASIC

A ← $FF
DP ← 0
X ← $6E00

BASIC・
COLD・START

※ DBA ： データバッファ
SCT ： セクタ（1～32）

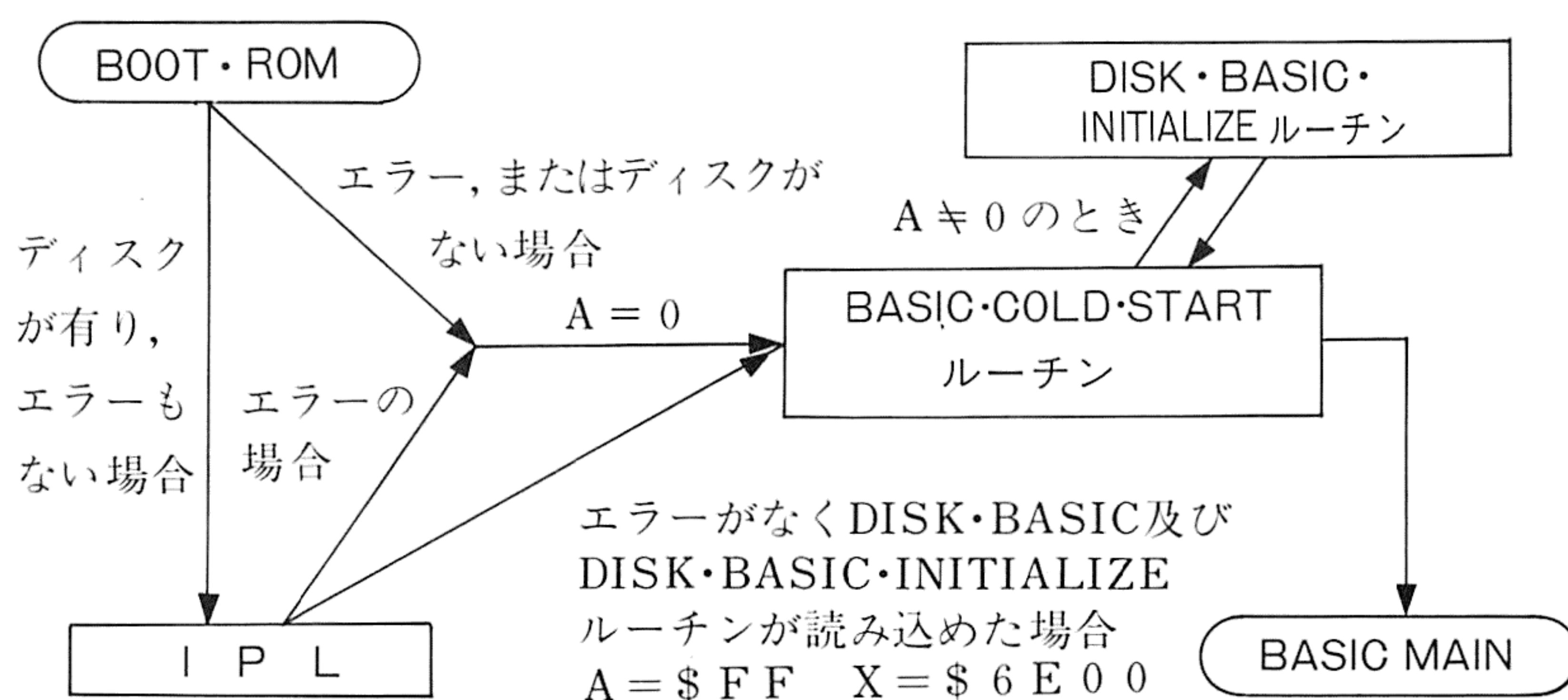
ＩＰＬのためのＲＣＢ群は＄０１０２番地から＄０１２１番地の間に格納されています．ただし，ＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥルーチンとＤＩＳＫ・ＢＡＳＩＣを設定するためのＲＣＢ（＄０１０２～と＄０１０Ａ～）では，途中でＲＣＢＤＢＡとＲＣＢＳＣＴが順次変化していきます．

ＩＰＬは，ＩＤセクタ内にある識別子によってＦ－ＢＡＳＩＣで使用可能なディスクであるかどうかを判断します．これについては，システム解説１３．９．７のＩＤの欄を御参照下さい．ここに違うコードが入っていた場合には，即座にＲＯＭモードになってしまいます．

ＩＰＬからＢＡＳＩＣへと制御が移る場合について多少説明を加えることにします．ＩＰＬの終了が訪れると（エラーで終わったにせよ，正当に終わったにせよ）一様にＣＯＬＤ・ＳＴＡＲＴのエントリ，すなわち＄８４８Ｂ番地〔＝（ＦＢＦＥ：ＦＢＦＦ）の内容〕にジャンプします．ここでＲＯＭ・ＢＡＳＩＣとＤＩＳＫ・ＢＡＳＩＣの違いは，ジャンプする間隙の唯一つの微妙な違いによって起こされるのです．

ＤＩＳＫ・ＢＡＳＩＣが読み込まれ，スタンバイの状態になっているときは，Ａレジスタが＄００以外の値を取り，Ｘレジスタには，ＤＩＳＫ・ＢＡＳＩＣの始まる番地が書かれています．ＲＯＭ・ＢＡＳＩＣが起動する際には，全てＡレジスタがクリアされています．

従って，この違いが，後にＣＯＬＤ・ＳＴＡＲＴルーチン上でＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥルーチンを呼ぶかどうかの違いになってきます．

BOOT・ROM
DISK・BASIC・INITIALIZE ルーチン
エラー，またはディスクがない場合
Ａ≒０のとき
ディスクが有り，エラーもない場合
Ａ＝０
BASIC・COLD・START ルーチン
エラーの場合
エラーがなくDISK・BASIC及びDISK・BASIC・INITIALIZEルーチンが読み込めた場合
Ａ＝＄ＦＦ　Ｘ＝＄６Ｅ００
I P L
BASIC MAIN

1―3　BASIC・COLD・STARTルーチン

ＢＯＯＴ・ＲＯＭから源を発したＢＡＳＩＣ起動への流れは，ブレーク・キーが押されていない限り，ＩＰＬを経る経ないによらず，ＢＡＳＩＣ・ＣＯＬＤ・ＳＴＡＲＴルーチンのエントリに辿り着きます．

ＢＡＳＩＣ・ＣＯＬＤ・ＳＴＡＲＴルーチンに入るためには，前述したように（ＦＢＦＥ：ＦＢＦＦ）上にジャンプベクトルが書かれていて，ＩＰＬやＢＯＯＴ・ＲＯＭは，このジャンプベクトルを頼りに，このエントリポイント（＝エントリのアドレス）に，処理を移していきます．このジャンプベクトルの値は＄８４８Ｂで，ここがエントリポイントとなり，＄８４８Ｂ以降に（～＄８７０２まで）ＣＯＬＤ・ＳＴＡＲＴルーチンがＲＯＭ上に展開しています．

1―3―1　主な動作

動作の内容をまとめると，次のようになります．

●Ａレジスタを判別し，その値が０以外のときは，前からきたＸレジスタの値(＝ＤＩＳＫ・ＢＡＳＩＣの先頭アドレス)を(ＦＦＦ８：ＦＦＦ９）に保存します．それ以外のときは，＄８０００が入ります．
●割り込みをマスクし，ＲＡＭ領域（ＲＯＭモードでは＄００００～＄７ＦＦＦ，ＤＩＳＫモードでは＄００００～＄６ＤＦＦの領域）を全てクリアします．
●ワークエリアを設定します．
●設定の途中で（ＦＣ００～ＦＣ７Ｆ）のＲＡＭ領域をクリアし，（ＦＦＦ８：ＦＦＦ９）に入っているＤＩＳＫ・ＢＡＳＩＣの先頭アドレスを（ＦＣ００：ＦＣ０１）に転送します．
●割り込みベクトルテーブルを設定します．
●ＢＩＯＳを初期化します．
●ＴＡＢを設定します（ＴＡＢキーの停止位置の設定）．
●ＰＦ・キーを設定します(ただし，ＤＩＳＫモードの場合は，ＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥルーチンが行います)．
●コンソール機能を設定します．
●タイマを設定します．
●画面を設定します．
●ＤＩＳＫモード〔(５９Ｆ)≒＄００〕であれば，(ＦＣ００：ＦＣ０１)を参照してＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥルーチンへジャンプします．

ＲＯＭモードの場合は，テキストを初期化した後，汎用読み込みポインタ（Ｄ９：ＤＡ）を＄８７５８（＄００が設定されている）に設定し，また，ダイレクトであることを指定〔（４７：４８）に＄ＦＦＦＦを代入〕します（ＤＩＳＫモードでも，同様のことを行っています）．

●最後に，（＄８６３Ｅ番地からで，ＤＩＳＫモードの場合でも，大抵はここに戻ってきますが）メッセージを表示し，割り込みマスクを解除した後，ＢＡＳＩＣインタプリタのメインルーチン（２章）へジャンプします．

BASIC・COLD・START
RAM領域のクリア
DISK・BASIC・INITIALIZE ルーチン
DISKモード
BIOSの初期化
TABの設定
PF・キーの設定
コンソール機能の設定
割り込みベクトルテーブルの設定
ワークエリアの設定
画面の設定
タイマの設定
メッセージの表示
※破線はサブルーチン

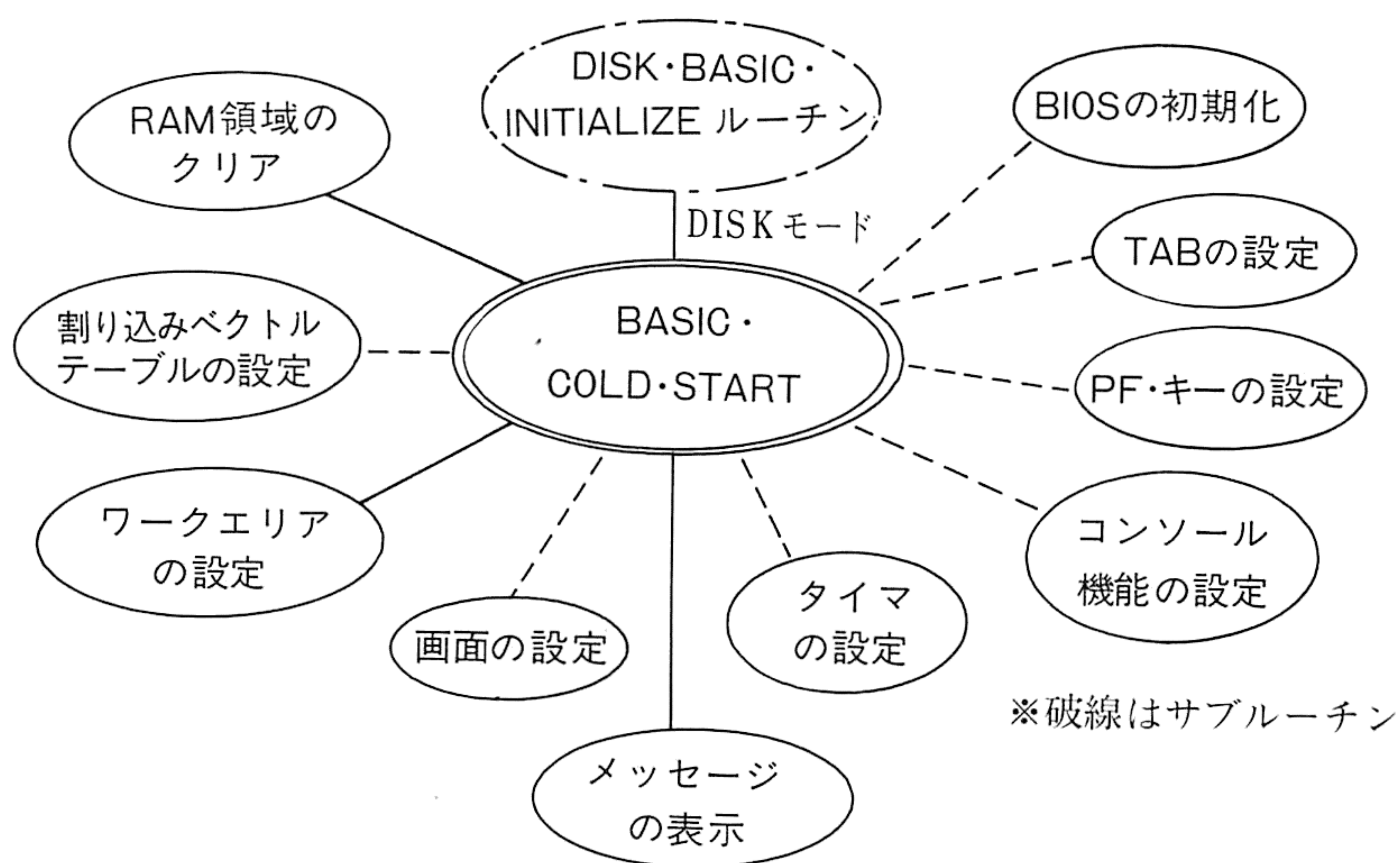



図1・3・1　ＢＡＳＩＣ・ＣＯＬＤ・ＳＴＡＲＴ　ルーチン

1―3―2　ワークエリアの設定

ＣＯＬＤ・ＳＴＡＲＴルーチンの主な仕事は，ワークエリアの初期設定です．ここでは，その中でも重要なものについて取りあげました．

(1)　**ＤＩＳＫ／ＲＯＭモード識別フラグ**

● （０５９Ｆ）　　　　：ＲＯＭ時…＄００　ＤＩＳＫ時…＄ＦＦ

(2)　**領域設定**

ＢＡＳＩＣの使用領域を設定します．

● （０５９Ｄ：０５９Ｅ）：解放ＲＡＭ領域の終わり

→ＲＯＭ時…＄７ＦＦＥ　ＤＩＳＫ時…＄７１Ｄ４
※文字列スタック領域先頭番地（４５：４６）も同じ値です．

- （００４１：００４２）：文字列使用領域境界（ＳＳＰ）
→ＲＯＭ時…＄７ＦＦＥ　ＤＩＳＫ時…＄６ＤＦＥ
※ＤＩＳＫ時は，後で＄７１Ｄ４に変更されます．
- （００３Ｆ：００４０）：文字列スタック領域最終番地
→ＲＯＭ時…＄７ＥＤ２　ＤＩＳＫ時…＄７０Ａ８
※スタックポインタも同じ値です．

(3) ＢＩＯＳジャンプルーチン

- （００ＤＥ～００Ｅ０）：→＄７Ｅ（ＪＭＰ），＄Ｆ１７Ｄ

(4) 入出力テーブルおよび割り込み時のテーブルのポインタ（４章参照）

入出力関係のバッファ，システムファイルディスクリプタ，割り込みのテーブルの先頭アドレス・ポインタを設定します．

- （０２Ｂ０：０２Ｂ１）：カセット用のバッファのポインタ→＄０５ＥＤ
- （０２Ｂ２：０２Ｂ３）：デバイス分類テーブルのアドレス→＄０６ＥＣ
- （０２Ｂ４：０２Ｂ５）：システムファイルディスクリプタの先頭アドレス→＄０７０Ｃ
- （０２Ｂ６：０２Ｂ７）：ＰＥＮ割り込みのテーブル（現在未使用）→＄０７１Ｅ
- （０２Ｂ８：０２Ｂ９）：ＣＯＭｎ割り込みのテーブル→＄０７２３
- （０２ＢＡ：０２ＢＢ）：ＫＥＹ割り込みのテーブル→＄０７３Ｃ
- （０２ＢＣ：０２ＢＤ）：ＴＩＭＥ割り込みのテーブル→＄０７６Ｅ
- （０２ＢＥ：０２ＢＦ）：ＩＮＴＥＲＶＡＬ割り込みのテーブル→＄０７７３
- （０５ＡＡ：０５ＡＢ）：プリンタの出力テーブル→＄０７７８

(5) プリンタ出力ジャンプテーブル　（４章参照）

- （０７７８～０７８Ｃ）：次のデータとジャンプアドレスが入ります．

→＋０～＋３　“ＬＰＴ０”　…デバイス名
＋４～　＄Ｄ９Ａ５，＄Ｄ６１５，＄ＣＥＦ４，
＄Ｄ６１７，＄Ｄ３９Ｃ，＄ＣＥＦ４，
＄ＣＥＦ４，＄００，＄５０，＄３８

(6) ＲＳ－２３２Ｃ入出力ジャンプテーブル

拡張モジュール（ここでは，ＲＳ－２３２Ｃをインタフェースすることが可能であるもの）が接続されているかどうかを，Ｉ／Ｏポート（＄ＦＤ０７，＄ＦＤ２５，＄ＦＤ２７，＄ＦＤ２９，＄ＦＤ２Ｂ）で調査して，接続されているものには，その入出力のサブルーチンへのジャンプ先（全て共通）を設定します．また，接続されているものだけをポート番号が小さい順に作っていくので，ワークエリア内のこれらのアドレスに何が入っているのかは，接続によることになります．

相対値	データとジャンプアドレス	ワークエリア
→＋０～＋３	“ＣＯＭｎ”…デバイス名（ｎ：ポート番号）	
＋４ ～ ＋１７	＄Ｄ１ＡＤ，＄Ｄ１９８， ＄Ｄ２Ｂ５，＄Ｄ２５Ｃ， ＄Ｄ３９Ｃ，＄Ｄ３Ａ５， ＄Ｄ３Ｂ４	（０７８Ｆ～０８２Ｄ） （０８２Ｅ～０８ＣＣ） （０８ＣＤ～０９６Ｂ） （０９６Ｃ～０Ａ０Ａ） （０Ａ０Ｂ～０ＡＡ９）
＋２３ ～ ＋２４	ＣＯＭ０…＄ＦＤ０６ それ以外…＄ＦＤ２２＋２＊ｎ（ｎ：ポート番号）	アドレスの小さいものへ，ポート番号が小さいものから設定されていく．ただし，接続されていないものがあると，設定されないものも出てくる．
＋２５	＄ＣＦ	

(7) ＲＳ－２３２Ｃポートのテーブルのポインタ

内容は(6)の先頭アドレスです．従って，接続されているポート番号の小さい方から設定されます．

- （０２Ｃ０：０２Ｃ１）：→＄０７８Ｆ
- （０２Ｃ２：０２Ｃ３）：→＄０８２Ｅ
- （０２Ｃ４：０２Ｃ５）：→＄０８ＣＤ
- （０２Ｃ６：０２Ｃ７）：→＄０９６Ｃ
- （０２Ｃ８：０２Ｃ９）：→＄０Ａ０Ｂ

(8) ＢＡＳＩＣテキストエリアの先頭アドレスの設定

テキストエリアの先頭アドレスは，ＲＳ－２３２Ｃのポート数によって変わります（また，ＤＩＳＫモード時は，加えてドライブ数とファイルバッファ数によっても変わってきます）．以下に，ＲＯＭモード時の値を示します．

● （0033：0034）：RS－232Cのポートの個数で異なります.

0個のとき　→$0790　　3個のとき　→$096D
1個のとき　→$082F　　4個のとき　→$0A0C
2個のとき　→$08CE　　5個のとき　→$0AAB

例　RS－232Cのポート0，2，3がつながっている場合

プリンタ	ポート0	ポート2	ポート3	BASICのテキスト or (DISKのバッファ)

778　78F　82E　8CD　96D

（0033：0034）　テキストエリアの先頭アドレス　$096D
（02C0：02C1）　ポート0のテーブルのポインタ　$078F
（02C2：02C3）　ポート2のテーブルのポインタ　$082E
（02C4：02C5）　ポート3のテーブルのポインタ　$08CD

(9) 解読実行ルーチン用テーブル（01EC～0216）の設定

キーワードなどのテーブルのポインタを設定します（他のものは2章参照）.

● （01EC～01EE）："ON～" の場合のエントリへのジャンプ命令
　→$7E（JMP），$DCA6

(10) マシン語サブルーチン，ユーザ関数の開始アドレスの初期設定

（初期設定では，全てIllegal Function Callエラーのエントリのアドレス＝$9663を指しています）

● （0217：0218）：EXECの開始アドレス　→$9663
● （0219：021A）：USR0の開始アドレス　→$9663
　　～
● （022B：022C）：USR9の開始アドレス　→$9663

(11) IRQ割り込みのサブルーチンのジャンプ命令の設定

● （05DF～05E1）：タイマの割り込みのジャンプ命令
　（PLAYの実行をします）
　→$7E（JMP），$ED72
● （05E2～05E4）：KEY・INの割り込みのジャンプ命令
　→$39（RTS）

(12) BASIC拡張用のサブルーチン・ジャンプ命令テーブル

BASIC内のメインルーチンや解読実行ルーチンなど基本的に重要なルー

チンの中では，このテーブルをサブルーチンとして呼んでいるものがあります．これは，将来ＢＡＳＩＣの拡張をする際の設定や，デバッグなどのために使うことができるようにするためです〔現在の時点ではＤＩＳＫモード時に一部のものが設定されていますが，それ以外は全て，＄３９（ＲＴＳ）が最初に入っており，ただ戻るだけとなっています．ＢＡＳＩＣに更に拡張した機能を持たせたい場合には，ここをジャンプ命令に変更して，ユーザの作ったサブルーチンにジャンプさせて下さい〕．

- （０２６０～０２６２）：ブロック入力ルーチン拡張用（＄ＤＡＡ６より）
- （０２６３～０２６５）：解読ルーチン拡張用（ＳＣ６３Ｄより）
- （０２６６～０２６８）：解読ルーチン拡張用（＄Ｃ６Ｄ８より）

∫

- （０２９Ｆ～０２９１）：テキスト作成ルーチン拡張用（＄Ｃ５０２より）

※他のものは，６－１節を御参照下さい．

1—3—3　サブルーチン概説

（1）ブロック転送ルーチン

アドレス　＄８５９７～＄８５９Ｅ

機　能　Ｘレジスタの示すアドレスから，Ｂレジスタのバイト数分だけ，Ｕレジスタの示すアドレス以降に転送する．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　Ｂレジスタ＝転送するバイト数（≦＄ＦＦ）
Ｘレジスタ＝転送元の先頭アドレス
Ｕレジスタ＝転送先の先頭アドレス

復帰情報　Ｂレジスタが０になるので，Ｚフラグがセットされる．
Ｘレジスタ＝転送元の終了アドレス＋１
Ｕレジスタ＝転送先の終了アドレス＋１

解　説　このルーチンを使う際，注意しなければならないのはＸ，Ｕレジスタの大小関係と転送領域の重なりです．Ｘ≧Ｕのときは問題はありませんが，Ｘ＜Ｕのときに，転送領域が重なる場合は，転送元のデータが消去されます．
なお，Ｂレジスタに＄００を入れて呼び出すと，２５６バイト分の転送が行われます．

(2) A，Xレジスタ転送ルーチン

アドレス　$859F～$85A6

機　能　A，Xレジスタの3バイトの内容を，Bレジスタの値の回数分メモリに繰り返し転送する．

レジスタ　A，B，X，U　（A，Xレジスタの値は保存される）

入力情報　A，Xレジスタ＝転送するべき値
Bレジスタ＝転送回数
Uレジスタ＝転送先の先頭アドレス

(3) BIOSイニシァライズルーチン

アドレス　$867B～$8683

機　能　BIOSをイニシァライズ（初期化）する．

レジスタ　A，X

WORK　(05A0)＝RCB用　※RCBについては，システム仕様参照

SUB　$00DE→BIOSジャンプルーチン

(4) 割り込みベクトル設定ルーチン

アドレス　$8692～$86A5

機　能　(FFF2～FFFD)の割り込みベクトルを設定する．

レジスタ　A，B，X，U

WORK　(FFF2～FFFD)＝割り込みベクトルレジスタ
(05D2)　＝割り込み（IRQ）マスクレジスタ
(FD02)　＝割り込み（IRQ）マスクI／Oポート

SUB　$8597→ブロック転送ルーチン

解　説　次の表のように，割り込みベクトルを設定します．設定後は，I／Oポート(FD02)の割り込みマスクのうち，RS－232CのRxRDYの割り込みだけイネーブルにします．(FD02）を設定した内容（＝$40）と同じものが（05D2）に保存されます．

割り込み名	割り込みレジスタ	割り込みベクトル	割り込みの応答
SWI3	FFF2:FFF3	01D1	RTI
SWI2	FFF4:FFF5	01D4	RTI
IRQ	FFF6:FFF7	01E0	JMP $D2FC
FIRQ	FFF8:FFF9	01DD	JMP $C953
SWI	FFFA:FFFB	01D7	RTI
NMI	FFFC:FFFD	01DA	RTI

(5) コンソール初期化ルーチン

アドレス $8656〜$867A

機能 種々のコンソール制御の初期化を行う.

レジスタ A, B, X, U

WORK (02A2〜02AB) =TAB設定テーブル
(00BF) =ファイル番号

SUB $86CC → PF・キー設定ルーチン
$86A6 → TAB設定ルーチン
$DBE1 → コンソール機能設定ルーチン
$E44E → タイマ設定ルーチン
$C8BC → 画面設定ルーチン
$9BDB → PRINT OUTルーチン

解説 簡単に各項目について説明します.

● PF・キーの設定 ($86CC)
$86CCルーチンは, 10個のPF・キーを一度に設定します.

● TAB・キーの停止位置の設定 ($86A6)
TAB設定テーブル(02A2〜02AB)をセットし, $86A6でBIOSのTAB SET〔システム仕様2.2.11(11)〕を呼び, 設定します.

● コンソール機能の設定
$DBE1を呼び, BIOSのCONSOLE CONTROL〔システム仕様2.2.11(12)〕をリクエストさせて, 制御フラグに$17を代入しているため, コンソール機能として, TAB動作, オーダ動作, カーソル表示, プットウェイト (ESC・キーの入力受付を可能にする) の処理を行うことになります.

●タイマの設定

＄E44Eは，BIOSのSET TIMER〔システム仕様2.2.14(1)〕をリクエストし，制御レジスタ（TC）に，＄08を代入して，0時割り込みのみを可能にします.

●画面の設定

＄C8BCでは，BIOSのINIT〔システム仕様2.2.11(1)〕をリクエストして画面の設定を行っています．このルーチンは，ワークエリアからパラメータを次のように取ります.

●背景色	＄00	（黒）	←	(01E6)
●桁数	＄28	(=40)	←	(00C3)
●行数	＄14	(=20)	←	(030D)
●スクロール開始行	＄00	(= 0)	←	(030E)
●スクロール終了行	＄13	(=19)	←	※
●PF・キー表示	＄00	(NO)	←	(0310)
●初期設定後の画面消去	＄01	(YES)		
●単色表示	＄00	(NO)	←	(05AD)

※ 次式で計算されます：(030F) − (030E) ＋1

●先行入力の禁止 〔システム仕様 2.2.4(8)〕

＄9BDBで（00BF）を使用するため，デフォルト値（=0）を設定して画面への出力指定をしています〔4章参照).次に，画面へ＄1B，＄68を＄9BDBを使って出力します（実際には，サブシステムに出力され，先行入力の禁止をオーダします．これを解除するときは，PRINT CHR＄（&H1B）＋CHR＄（&H67）を実行します〕.

※ TAB設定ルーチン（＄86A6),PF・キー設定ルーチン（＄86CC，＄86CF）については，フェーズIを御参照下さい.

1—4 DISK・BASIC・INITIALIZE

ＤＩＳＫモード時に起動されるこのルーチンは，その役目が済んだ後は消去される運命にあります．しかし，その間に行う初期設定は，ＤＩＳＫ・ＢＡＳＩＣを呼ぶ際に必要不可欠のものです．

ＤＩＳＫ・ＢＡＳＩＣ・ＩＮＩＴＩＡＬＩＺＥ（以下**ＤＢＳＩＮＩ**と略す）ルーチンは，データも含め（６Ｅ００～７１Ｄ５）に配置されており，（０５９Ｆ）の内容が＄００でないとき（ＦＣ００：ＦＣ０１）の示すアドレス（＝＄６Ｅ００）に従って起動されます．このルーチンは，また，ユーザのプログラム（ＢＡＳＩＣで記述されているもの）のオート・スタートを行うこともできます．なお，このルーチンの（６Ｅ００～６ＦＦＦ）の部分は，トラック０のセクタ１５,１６に格納されています．

1—4—1　主な動作

- ディスクからＩＤセクタを読み，そこにドライブ数とファイルバッファ数が書かれていれば，それらを設定します．
- ディスクのディレクトリで“ＳＴＡＲＴＵＰ”というファイルの有無を調べ，ない場合はドライブ数とファイル数をキーボードから入力し，設定します．
- ワークエリアの設定を行います．
  （１）ディスク用のバッファ領域の確保
  （２）ディスク用に新たに追加されたキーワード，関数などの更新処理
  （３）ディスク用の入出力サブルーチンのジャンプ命令の設定
  （４）解放ＲＡＭ領域の確保
- テキスト・変数の消去，初期化を行います．
- “ＳＴＡＲＴＵＰ”というファイルがあれば，それを，ＲＵＮ（オート・スタート）させます．ない場合は，ＣＯＬＤ・ＳＴＡＲＴルーチンに戻ります．

1—4—2　ユーザ・プログラムのオート・スタート

オート・スタートが可能な条件は，次の２つです．

- ＩＤセクタの３，４バイト目に，各々ドライブ数（＄０１～＄０４）とファイルバッファ数（＄００～＄０Ｆ）が書かれていること．
- “ＳＴＡＲＴＵＰ”というファイル名でセーブ（ＳＡＶＥ）された，プログラム（ＢＡＳＩＣで記述）がドライブ０にあること．

図1・4・1　DBSINIの主な動作（オート・スタート関係を中心に）

$6E00

DBSINI

DBA=$7213
TRK=0
SCT=3

DISK READ

ERROR？　有　無

IDセクタを読み込み，ドライブ数とファイル・バッファ数が書かれていれば，それらを設定する

IDセクタからドライブ数・ファイルバッファ数を読み込む

IDセクタに書かれていたか　NO　YES

ドライブ数・ファイルバッファ数の設定

TRK=1
SCT=4

SCT=33?　YES　NO

DISK READ

ディレクトリから，"STARTUP"という名のファイルを見つけ出す

ディレクトリはまだあるか　NO　YES

ファイル名="STARTUP"？　YES　NO

次のファイル名にする

そのセクタの終り？　NO　YES

SCT←SCT+1

AT・ST=TRUE

AT・ST=FALSE

オート・スタートでない場合は，キーボードから入力を行う

キーボードから，ドライブ数・ファイルバッファ数を読み込む

ドライブ数，ファイルバッファ数の設定

ワークエリアの設定，およびその他の処理

ユーザのプログラムを，オート・スタートさせることができる

AT・ST？　FALSE　TRUE

RUN "STARTUP" を実行する

USER PROGRAMへ

COLD・STARTへ

※DBA　：データバッファの先頭番地
TRK　：トラック（0～39）
SCT　：セクタ　（1～32）
AT・ST：オート・スタート　フラグ（7213）
DISK READ：4－5－4 (4) 参照

そこで，サンプルとして，ＩＤセクタの３バイト目と４バイト目を設定するプログラムとオート・スタートで起動するプログラムをＢＡＳＩＣで作りました．

１番目のプログラムで，ＩＤセクタを書き込み，２番目のプログラムを入力して，“ＳＴＡＲＴＵＰ”というファイル名でセーブして下さい．

(SAVE "STARTUP" ⏎)

リセットボタンを押して，ＣＯＬＤ・ＳＴＡＲＴさせると，２番目のプログラムがオート・スタートします．

```
100 ' ID WRITER for AUTO START
110 ' ****************************************************
120 '   DRV    .....  Number of Drives
130 '   FBUF   .....  Number of File Buffers
140 '   DRV$   .....  CHR$(DRV)  for ID sector write
150 '   FBUF$  .....  CHR$(FBUF) for ID sector write
160 '   A$,B$  .....  Variables for System Random Buffer
170 ' ****************************************************
180 CLEAR 1000
190 ' INPUT Drives & File Buffers
200   INPUT "How many Drives do you want";DRV
210     IF DRV>4 OR DRV<1 THEN BEEP:GOTO 200
220   INPUT "How many File Buffers do you want";FBUF
230     IF FBUF>15 OR FBUF<0 THEN BEEP:GOTO 220
240 ' CHANGE integer to character for Sector Write
250   DRV$=CHR$(DRV)
260   FBUF$=CHR$(FBUF)
270 ' WRITE ID Sector
280   FIELD #0,128 AS A$,128 AS B$
290    ' read
300    DUMMY$=DSKI$(0,0,3)
310   ' change
320    LSET A$=LEFT$(A$,3)+DRV$+FBUF$+RIGHT$(A$,123)
330   ' write
340    DSKO$ 0,0,3
350 END
```

```
RUN
How many Drives do you want? 2
How many File Buffers do you want? 5

Ready
```

```
DISK EDITOR FOR FM-7
          DRIVE : 0   TRACK : 0    SECTOR : 3
     +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00 : 53 00 00 02 05 00 00 00 00 00 00 00 00 00 00 00   :S...............
10 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
20 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
40 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
80 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
90 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
A0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
B0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
C0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
D0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
E0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   :................
KEY 'Q' - ESCAPE THIS MODE  'L' - HARDCOPY
```

```
10 ' START UP PROGRAM
20 '  *******************************************************
30 '     FNA(X)   ........  Get 2-Byte Data from Memory(X:X+1)
40 '     AD       ........  Device Table Address Pointer
50 '     DV$      ........  Device Name (get from Device Table)
60 '     K        ........  Sentinel for printing "I/O"
70 '    WORK・AREA
80 '      (71FA)  ........  Number of Drives
90 '      (71FB)  ........  Number of File Buffers (for User)
100 '     (0033:0034)  ...  Free Area Top Address
110 '     (003F:0040)  ...  Free Area End Address
120 '     (06EC-    )  ...  Device Table Address
130 ' *******************************************************
135 '
140 DEF FNA(X)=CVI(CHR$(PEEK(X))+CHR$(PEEK(X+1)))
150 '
160 WIDTH 80
170 ' Print out Messages
180 PRINT " USER DISK BASIC STARTS":PRINT
190 PRINT "Message :";
200 PRINT SPC(3)"You can access next Devices."
210 ' Print out usable DISK Drives
220 PRINT SPC(16);"Disks..."
230   FOR I=0 TO PEEK(&H71FA)
240     PRINT SPC(22);STR$(I);":";SPC(5);
250      K=3:GOSUB 490
260    NEXT
270 ' Print out usable Inter Devices
280 PRINT SPC(16);"Others.."
290   AD=&H6EC:K=1
300   WHILE FNA(AD)<>0
310     DV$=""
320     FOR I= 0 TO 3
330       DV$=DV$+CHR$(PEEK(FNA(AD)+I))
331      'Get character from Device Table
340     NEXT
350     PRINT SPC(20);DV$;":";SPC(5);
360     GOSUB 490
370     AD=AD+2:K=K+1
380   WEND
385 ' Print out other messages
390 PRINT
400   PRINT TAB(12)"You have ";PEEK(&H71FB);"File Buffers."
410 PRINT
420   PRINT TAB(12) "Free Text Area Top is &H";
430   PRINT RIGHT$("0"+HEX$(FNA(&H0033)),4);  "."
440 PRINT
450   PRINT TAB(12) "Free Area Size is &H";
460   PRINT HEX$(FNA(&H003F)-FNA(&H0033));  " Bytes."
470 NEW
480 END
485 ' Print out Input or Output
490  ON K GOTO 510,520,500,520
500    PRINT "For Input & Output":RETURN
510    PRINT "For Input"         :RETURN
520    PRINT "For Output"        :RETURN
530  END
```

```
 USER DISK BASIC STARTS

Message :    You can access next Devices.
                 Disks...
                        0:      For Input & Output
                        1:      For Input & Output
                 Others..
                     KYBD:      For Input
                     SCRN:      For Output
                     CASO:      For Input & Output
                     LPTO:      For Output

            You have  5 File Buffers.

            Free Text Area Top is &H1035.

            Free Area Size is &H6073 Bytes.

Ready
```

1—4—3　ドライブ数，ファイルバッファ数の決定

オート・スタートする場合は，ＩＤセクタから読み込んで設定されますが，それ以外の一般の場合はキーボードからユーザが入力することになります．
キーボードから入力する際の動作は，次のようになっています．

(1)“ＤＩＳＫ　ＶＥＲＳＩＯＮ”を出力します．

Ｘレジスタに文字データの先頭番地（＝＄７１９Ｃ）を入れ，ＰＲＩＮＴルーチン（＝＄７４Ａ６）で出力します．

(2)“How many disk drives ?”を出力します．

この出力は，“How many disk”と“drives ?”を別々に行っています．出力の仕方は前回と同様ですが，文字データの先頭番地は各々，＄７１ＡＢ，＄７１ＢＡとなります．また，これらの出力の前にフィールド設定ルーチン（＝＄Ｄ９２Ｆ）を呼んで，文字出力のフィールドの設定をしています．

(3)キー入力をして，ドライブ数を設定します．

まず，ＢＥＥＰルーチン（＝＄ＤＢ４９）を呼んでから，１行入力ルーチン（＝＄Ｄ８０７）を呼び，キーボードからドライブ数を入力します．この際，何も入力しないでＲＥＴＵＲＮキーを押すと，デフォルトとして２が設定されます．何か文字が入力された場合は，文字解読をして数値に変換するルーチン（＝＄７０５Ｃ）を呼んで，範囲内なら（１以上４以下)，その値に－１したものが（７１ＦＡ）に入ります．

※－１するのは，ドライブ番号が０から始まっているからです．

(4)“How many disk files　(０—１５)　?”を出力します．

(2)と同様に，２つの部分に分けて出力しています．“files (０—１５)?”の文字データの先頭アドレスは＄７１Ｃ８です．文字データは，その最初のデータが文字列の長さを示し，その次から文字列データが長さ分入っています．＄７４Ａ６のＰＲＩＮＴルーチンのための文字データは，いつもこのようにしておきます．

(5)キー入力をして，ファイルバッファ数を設定します．

(3)とほぼ同様ですが，範囲は，０以上１５以下になります．また，数値は，（７１ＦＢ）に設定されます．

キー入力の際も，フィールド設定ルーチン（＝＄Ｄ９２Ｆ）を呼んでフィールドの設定をしています．

1—4—4　DISKモードでのワークエリアの設定

ＤＩＳＫモードでは，新たにディスク用のバッファや，ワークエリアが設定されます．ここでは，その中で重要なものについて解説します．

(1)　ドライブ数，ファイルバッファ数の設定

ＩＤセクタ，あるいはキーボードからの入力により，設定されます．

● （７１ＦＡ）　：ドライブ数（０～３）
→入力した数－１の値がここに入る

● （７１ＦＢ）　：ファイルバッファ数（０～１５）
→入力した数の値が入る

(2)　ドライブテーブルの先頭アドレスの設定

各ディスクドライブの中に入っているディスクからＦＡＴ(File Allocation Table＝各クラスタの使用状況を記録する領域）を，読み取り，変更し，再び記入するための領域がドライブテーブルです．これは，ドライブ数（１～４）分，確保されます．また，その先頭アドレスは，ＲＳ―２３２Ｃが何個装着されているかによって異なってきます．ここでは，０個と１個のときの値を示しますが，それ以外のときは，計算法に従い計算して下さい．

● （７１Ｄ６：７１Ｄ７）：ドライブテーブルの先頭アドレス
→０個…＄０７８Ｆ　１個…＄０８２Ｅ

○計算法
＄０７８Ｆ＋｛（ＲＳ―２３２Ｃの数)＊＄９Ｆ｝

(3)　ファイルバッファの先頭アドレスの設定

ディスクに対して割り当てられるファイル用のバッファ領域が取られます．これらは，ファイルバッファ数＋２個分，先頭アドレスが設定されます．２個分余計に取られる理由は，ＤＩＳＫ・ＢＡＳＩＣで使う「ＦＩＥＬＤ　＃０，～」で設定される「ＤＳＫＯ＄」と「ＤＳＫＩ＄」用（システムランダムバッファ）のものが一つと，プログラムの入出力や「ＤＳＫＩＮＩ」(システム用）のもので計２個分余分に取られます．ファイルバッファの先頭アドレスは，場合によって値が変わってきますの

で計算法のみを載せておきます．実際には，**表1・4・1**にRS－232Cが0個のとき，**表1・4・2**にRS－232Cが1個のときの値が出ています（なお，ファイル番号とファイルバッファ番号は，一致しないので注意して下さい）．

- （71D8：71D9）：システムランダムバッファの先頭アドレス
- （71DA：71DB）：ファイルバッファ＃＃0（システム用）の先頭アドレス
- （71DC：71DD）：ファイルバッファ＃＃1の先頭アドレス

  ※（71FB）＋2個分だけ，その先頭アドレスが設定される．それ以外のものは，$0000の値になっている．

  ⌠

- （71F6：71F7）：ファイルバッファ＃＃14の先頭アドレス
- （71F8：71F9）：ファイルバッファ＃＃15の先頭アドレス

○計算法

ファイルバッファ＃＃Xの先頭アドレス
＝（71D6：71D7）＋｛（ドライブ数）＊$9E｝
　　　　　　　　＋｛（X＋1）＊$010F｝

※Xはファイルバッファ番号を示す．ドライブ数は，1～4の範囲である．

(4) BASICテキストエリアの先頭アドレスの設定

ROMモードでは，テキストの先頭アドレスは，RS—232Cの個数によって変化しましたが，DISKモードの時はドライブ数，ファイルバッファ数によっても変化してきます．具体値は**表1・4・1**や**表1・4・2**に示します．

- （0033：0034）：テキストの先頭アドレス

○計算法

テキストの先頭アドレス
＝$078F＋｛（RS—232Cの数）＊$9F｝
　　　　＋｛（ドライブ数）＊$9E｝
　　　　＋｛（ファイルバッファ数＋2）＊$010F｝＋1

※ドライブ数は，1～4の範囲で，ファイルバッファ数は，0～15の範囲です．

例1　ＲＳ－２３２Ｃ　０；ドライブ数１；ファイルバッファ数０

ドライブテーブル	システムランダムバッファ	ファイルバッファ ＃＃０	ＢＡＳＩＣのテキスト領域
078F	082D	093C	0A4C

※システムランダムバッファは「ＦＩＥＬＤ＃０～」用に，ファイルバッファ＃＃０はプログラムの入出力用などシステムで使われるため，ユーザ用のファイルバッファがなくユーザのファイルをディスクに対してＯＰＥＮすることはできない．

例2　ＲＳ－２３２Ｃ（ポート０）１；ドライブ数２；ファイルバッファ数５

ポート０バッファ	ドライブテーブル	ドライブテーブル	システムランダムバッファ	ファイルバッファ ＃＃０	…	…	ファイルバッファ ＃＃５	テキスト領域
078F	082E		09DB	0AEA	0BF9	0F26	1036	

表1・4・1　ドライブテーブルとファイルバッファとテキストの先頭アドレス

DRIVE TABLE TOP ADDRESS & FILE BUFFER TOP ADDRESS

RS232C = 0

DRIVES		1	2	3	4
(71D6:D7)	DRVTBL	078F	078F	078F	078F
(71D8:D9)	SYSRND	082D	08CB	0969	0A07
(71DA:DB)	## 0	093C	09DA	0A78	0B16
(71DC:DD)	## 1	0A4B	0AE9	0B87	0C25
(71DE:DF)	## 2	0B5A	0BF8	0C96	0D34
(71E0:E1)	## 3	0C69	0D07	0DA5	0E43
(71E2:E3)	## 4	0D78	0E16	0EB4	0F52
(71E4:E5)	## 5	0E87	0F25	0FC3	1061
(71E6:E7)	## 6	0F96	1034	10D2	1170
(71E8:E9)	## 7	10A5	1143	11E1	127F
(71EA:EB)	## 8	11B4	1252	12F0	138E
(71EC:ED)	## 9	12C3	1361	13FF	149D
(71EE:EF)	##10	13D2	1470	150E	15AC
(71F0:F1)	##11	14E1	157F	161D	16BB
(71F2:F3)	##12	15F0	168E	172C	17CA
(71F4:F5)	##13	16FF	179D	183B	18D9
(71F6:F7)	##14	180E	18AC	194A	19E8
(71F8:F9)	##15	191D	19BB	1A59	1AF7

TEXT TOP
(0033:34)

	1	2	3	4
FILE BUFFERS = 0	0A4C	0AEA	0B88	0C26
FILE BUFFERS = 1	0B5B	0BF9	0C97	0D35
FILE BUFFERS = 2	0C6A	0D08	0DA6	0E44
FILE BUFFERS = 3	0D79	0E17	0EB5	0F53
FILE BUFFERS = 4	0E88	0F26	0FC4	1062
FILE BUFFERS = 5	0F97	1035	10D3	1171
FILE BUFFERS = 6	10A6	1144	11E2	1280
FILE BUFFERS = 7	11B5	1253	12F1	138F
FILE BUFFERS = 8	12C4	1362	1400	149E
FILE BUFFERS = 9	13D3	1471	150F	15AD
FILE BUFFERS = 10	14E2	1580	161E	16BC
FILE BUFFERS = 11	15F1	168F	172D	17CB
FILE BUFFERS = 12	1700	179E	183C	18DA
FILE BUFFERS = 13	180F	18AD	194B	19E9
FILE BUFFERS = 14	191E	19BC	1A5A	1AF8
FILE BUFFERS = 15	1A2D	1ACB	1B69	1C07

* DRVTBL = DRIVE TABLE
* SYSRND = SYSTEM RANDOM BUFFER
* ##0 used for SYSTEM
* The FILEBUFFER not set up by DBSINI takes $0000

表1・4・2　ドライブテーブルとファイルバッファとテキストの先頭アドレス

DRIVE TABLE TOP ADDRESS & FILE BUFFER TOP ADDRESS

RS232C = 1

DRIVES		1	2	3	4
(71D6:D7)	DRVTBL	082E	082E	082E	082E
(71D8:D9)	SYSRND	08CC	096A	0A08	0AA6
(71DA:DB)	## 0	09DB	0A79	0B17	0BB5
(71DC:DD)	## 1	0AEA	0B88	0C26	0CC4
(71DE:DF)	## 2	0BF9	0C97	0D35	0DD3
(71E0:E1)	## 3	0D08	0DA6	0E44	0EE2
(71E2:E3)	## 4	0E17	0EB5	0F53	0FF1
(71E4:E5)	## 5	0F26	0FC4	1062	1100
(71E6:E7)	## 6	1035	10D3	1171	120F
(71E8:E9)	## 7	1144	11E2	1280	131E
(71EA:EB)	## 8	1253	12F1	138F	142D
(71EC:ED)	## 9	1362	1400	149E	153C
(71EE:EF)	##10	1471	150F	15AD	164B
(71F0:F1)	##11	1580	161E	16BC	175A
(71F2:F3)	##12	168F	172D	17CB	1869
(71F4:F5)	##13	179E	183C	18DA	1978
(71F6:F7)	##14	18AD	194B	19E9	1A87
(71F8:F9)	##15	19BC	1A5A	1AF8	1B96

TEXT TOP
(0033:34)

	1	2	3	4
FILE BUFFERS = 0	0AEB	0B89	0C27	0CC5
FILE BUFFERS = 1	0BFA	0C98	0D36	0DD4
FILE BUFFERS = 2	0D09	0DA7	0E45	0EE3
FILE BUFFERS = 3	0E18	0EB6	0F54	0FF2
FILE BUFFERS = 4	0F27	0FC5	1063	1101
FILE BUFFERS = 5	1036	10D4	1172	1210
FILE BUFFERS = 6	1145	11E3	1281	131F
FILE BUFFERS = 7	1254	12F2	1390	142E
FILE BUFFERS = 8	1363	1401	149F	153D
FILE BUFFERS = 9	1472	1510	15AE	164C
FILE BUFFERS = 10	1581	161F	16BD	175B
FILE BUFFERS = 11	1690	172E	17CC	186A
FILE BUFFERS = 12	179F	183D	18DB	1979
FILE BUFFERS = 13	18AE	194C	19EA	1A88
FILE BUFFERS = 14	19BD	1A5B	1AF9	1B97
FILE BUFFERS = 15	1ACC	1B6A	1C08	1CA6

* DRVTBL = DRIVE TABLE
* SYSRND = SYSTEM RANDOM BUFFER
* ##0 used for SYSTEM
* The FILEBUFFER not set up by DBSINI takes $0000

(5) システムランダムバッファの初期設定

このバッファは，「ＦＩＥＬＤ♯０～」で設定される「ＤＳＫＯ＄」，「ＤＳＫＩ＄」用で，それ以外の用途で使うことがないので，ここでランダムファイルのバッファとしての初期設定を行っておきます.

（７１Ｄ８：７１Ｄ９）の値をＸとおくと次のようになります.

- （　Ｘ　）　：ランダムファイルの指定
  →＄４０
- （Ｘ＋０７：Ｘ＋０８）：ファイルバッファの長さ
  →＄０１００　（２５６バイト）
- （Ｘ＋０９：Ｘ＋０Ａ）：Ｉ／Ｏバッファの始まるアドレス
  →Ｘ＋＄０Ｆ

1—4—5　ワークエリア設定後の動作

ワークエリア設定後のＤＢＳＩＮＩの動作は，以下のようになります．

●＄８Ｆ３９をコールして，テキスト，変数の消去と初期化を行います．

●汎用読み込みポインタ(Ｄ９：ＤＡ)を＄８７５８にします．

●現在，ダイレクトモードであることを示します〔(００４７：００４８)を，＄ＦＦＦＦにします〕．

※以上の３つは，ＣＯＬＤ・ＳＴＡＲＴルーチンで行っていることですが，その設定する前に，このルーチンにジャンプしてきたため，代りにここで行っています．

●Ｐ・ＦキーをＤＩＳＫモード用にします．具体的には，Ｘレジスタに＄６Ｅ０３を代入し，＄８６ＣＦを呼び出します．ＰＦ･キーのデータは，＄６Ｅ０３～＄６Ｅ４３に格納されています．

これらの処理が終わると，オート･スタートフラグ（７２１３）を参照して，その値が０ならば，割り込みマスクを解除して，＄８６３Ｅ（１－３－１節）へジャンプします．オート･スタートフラグ（７２１３）の値が０以外であれば，割り込みマスク解除後，ＲＵＮ“ＳＴＡＲＴＵＰ”を実行します．

1—5　HOT・START

ＨＯＴ・ＳＴＡＲＴルーチンは，ＲＥＳＥＴがかかったときに，ブレーク・キーが押されていれば起動します．そのエントリは，(ＦＢＦＣ：ＦＢＦＤ)の内容，すなわち＄８６８４です（１—１—２節参照)．

ＨＯＴ・ＳＴＡＲＴルーチンの動作は，以下のようになります．

●ＤＰレジスタをクリアします．

●コンソールなどの初期化を行います（＄８６５６をコールします)．

※このため，ホット・スタートをかけると，ＰＦ･キーは，ＤＩＳＫモード時でも，ＲＯＭモード時と同様に設定されます（詳しくは，１—３－３節の＄８６５６を御参照下さい．以下同様)．

●ＢＩＯＳをイニシァライズします（＄８６７Ｂをコールします)．

●割り込みベクトルを設定します（＄８６９２をコールします)．

●割り込みマスクを解除します（ＣＣレジスタのＦフラグとＩフラグを０にします)．

これらの処理が終わった後，Abort ルーチン（＄Ｄ６８０）に飛びます．

第2章　BASICインタプリタの骨格

はじめに

ＦＭ－７をはじめ，現在普及しているパソコンはすべて，ＢＡＳＩＣ(Beginner's All Purpose Symbolic Instruction Code) と呼ばれるプログラミング言語体系が主軸となって構成されています．しかし，ＢＡＳＩＣで書かれたプログラムを実際に実行しているのは，インタプリタと呼ばれる，おびただしい数のマシン語ルーチンの有機的結合体なのです．なお，「ルーチン」というのは「一定の仕事を行う，小さなプログラムのひと塊」のことをいいます．

さて，もうすでに御存知の方も多いかと思いますが，我々がキーボードを叩いて打ち込んだ「ＰＲＩＮＴ　Ａ」とか「Ｙ＝ＳＩＮ（Ｘ)」などといったプログラムは，そのままアスキーコードでメモリーに書き込まれるのではなく，中間コードと呼ばれる一種の暗号に変換された形でメモリーに格納されます．中間コードというのは，我々が目で見てわかる「ＰＲＩＮＴ　Ａ」などといったＢＡＳＩＣ言語と，コンピュータが即実行することのできる「７Ｅ　ＡＢ　Ｆ４（＄ＡＢＦ４番地へとジャンプする命令コード)」のようなマシン語との合の子的な存在であって，インタプリタの骨格を成しているのは，中間コードの集合体である中間言語プログラムを作成する部分と，中間言語プログラムを実行する部分の２つであるといってよいでしょう．多くの，ＢＡＳＩＣをある程度マスターした方々にとって，ＢＡＳＩＣプログラムが中間言語へと変換されていく過程，また，中間言語として蓄えられているプログラムがどのように解読され実行されていくのか，ということは，興味津々な事柄であるにもかかわらず，謎のベールに包まれた神秘の森で，なかなか立ち入り難いところではないでしょうか．実際，インタプリタについて，詳しく知り尽くすためには，何十 Kbyte ものマシン語プログラムを解析しなければならず，かなり骨の折れる仕事であるといえましょう．

この章では，主として，ＢＡＳＩＣプログラムを中間コードに翻訳するルーチンと，中間コードを解読・実行するルーチンの２つの部分にメスを入れ，ＢＡＳＩＣ言語体系の骨格となる部分について謎解きをしていきます．

なお，「ＢＡＳＩＣプログラムの地の文」という表現がいくつか出てきますが，「地の文」とは主に「注釈文字列や文字列定数でない部分」のことを指していいます．

2—1　ＢＡＳＩＣプログラムの格納形式

ＢＡＳＩＣのプログラムを作成するにあたって，重要な領域が３つあります．行入力バッファ（＄０４３Ｄ番地～），テキストバッファ（＄０３３Ｃ番地～），テキストエリア（標準ＲＯＭモード時，＄０７９０～）の３つです．また，各エリアがどこから始まってどこで終わるのかを指し示すポインタ群をはじめ，各種の重要な情報が詰め込まれているワークエリア（＄００００～＄０７８Ｆ番地）は，常時インタプリタから参照されており，先の行入力バッファ，テキストバッファはともに，このワークエリア内にあります（図２・１・２参照）．

さて，我々がキーボードを叩いて入力したプログラムは一行入力ルーチンによって，アスキーコードのまま「行入力バッファ」に一時的に蓄えられます．行入力バッファ内の１行のプログラムは，次の瞬間「ＢＡＳＩＣ⇨中間言語翻訳ルーチン」によって中間言語に翻訳されながら，「テキストバッファ」に格納されていきます．最後に，入力された１行が「行番号」を持っていれば，テキストバッファ内の１行が「テキストエリア」内の所定の位置に転送され，「行番号」を持っていなければ，「ダイレクト命令」とみなして実行されます．以上の様子を簡単に図示したのが図２・１・１です．

図２・１・１　一行入力および翻訳・格納の様子

キーボード
一行入力ルーチン
行入力バッファ
（４３Ｄ～５３Ｃ）
ＢＡＳＩＣ→中間言語
翻訳ルーチン
（一行翻訳の実行）
テキストバッファ
（３３Ｃ～４３Ｂ）
行番号があれば一行格納
行番号が
なければ
ダイレクト実行
テキストエリア

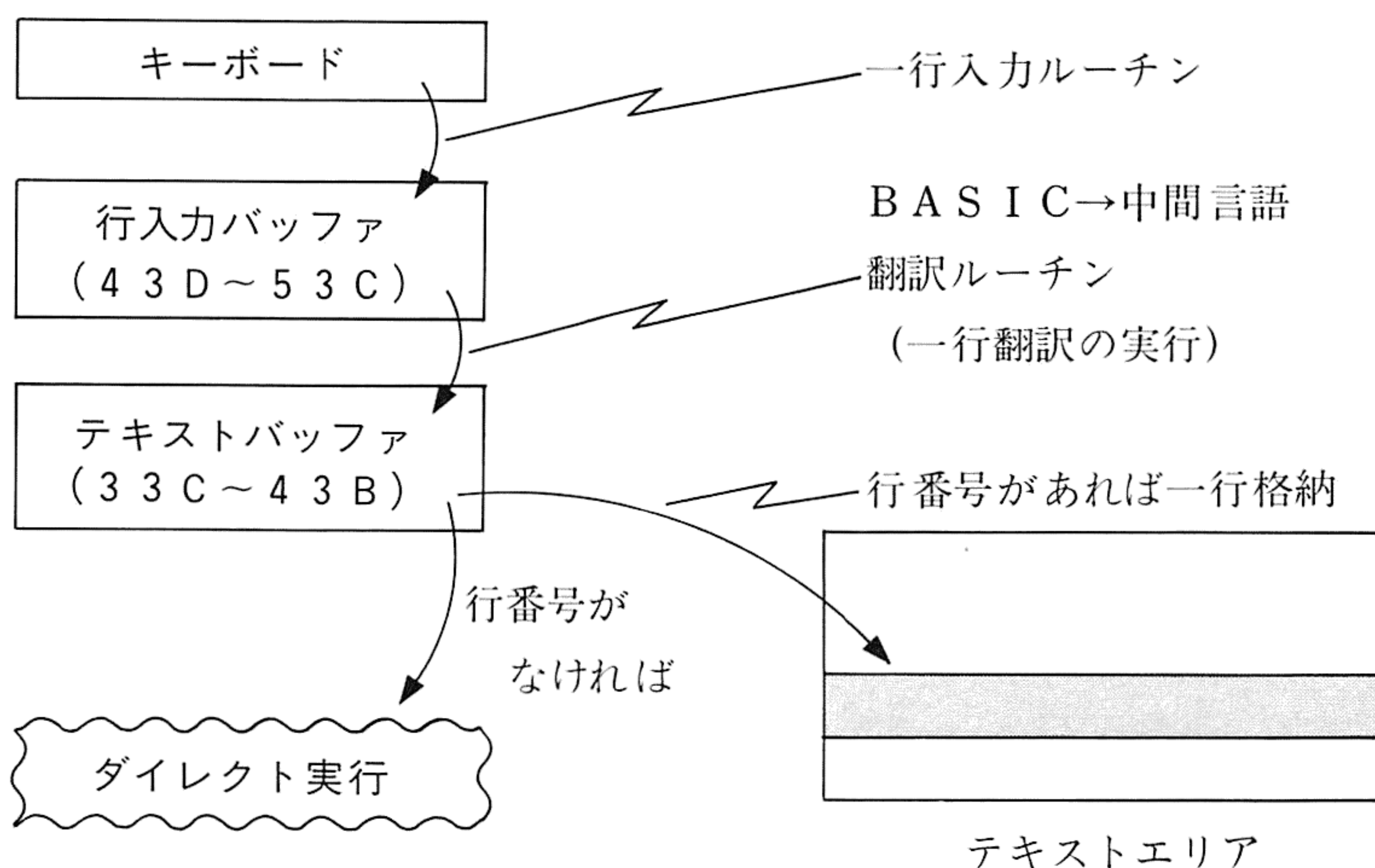



図２・１・２　ＲＡＭ領域のメモリーマップとワークエリア（ＤＩＳＫモード）

アドレス		主要なポインタ.
0000 〜 078F	ワークエリア	
0790	ドライブテーブル & ファイルバッファ等	
(0D08) [2ドライブ 2ファイルの DISK時]	↓ ↓ ↓ テキストエリア [BASICの中間言語が格納される]	←（33：34）
	↓ ↓ ↓ 単純変数の格納エリア	←（35：36）
	↓ ↓ ↓ 配列変数の格納エリア	←（3B：3C）
	↓ ↓ ↓ フリーエリア スタック領域 ↑ ↑ ↑	←（3D：3E）
	＄00	←（AF：B0）
空白の1バイト…	//////	←（3F：40）
	文字列領域 ↑ ↑ ↑	←（45：46）
71D4	ユーザーズエリア	←（59D：59E）
71D5 〜 7FFF	DISKコード	
8000	インタプリタ ROM	

アドレス	
0000	
13:14	テキストバッファ中の文の長さ（バイト数）＋5
33:34	テキストエリア先頭番地
35:36	単純変数の先頭番地
3B:3C	配列変数の先頭番地
3D:3E	フリーエリア先頭番地
3F:40	文字列領域の終わり
45:46	文字列領域の始まり
47:48	現在実行中の行番号
49:4A	ＣＯＮＴ再スタート行番号
4B:4C	サブルーチン呼出時の受渡用
4D:4E	CONT再スタート・テキスト番地
4F:50	現在実行中のテキスト番地
D9:DA	汎用読込みポインタ
33A 〜 33B	現在編集中の行番号（33C〜43Bの一行に対応）
33C 〜 43B	↓ ↓ ↓ テキスト作成用バッファ（テキストバッファ）
43D 〜 53C	↓ ↓ ↓ 行入力バッファ

それでは，中間言語プログラムは一体どんな形で格納されるのでしょうか．
一例として，まず，

```
10␣PRINT␣A
20␣GOTO␣20
```

といった，簡単なプログラムを書き込んで調べてみましょう．

このプログラムが，広いメモリ空間のどこに書き込まれているかを調べるために，MON（モニタ）コマンドを入れて，モニタ上で「D0033」と打ち込んでみると，（33：34）は$0790（この値は標準ROMモードでの値であり，DISKモードで例えばDRIVE数2，FILE数2を指定すれば，$0D08となります）となっています．図2・2・2に示した通り，（33：34）にはテキストエリア先頭番地が入っていますから，BASICの中間言語は，$0790番地から格納されていることがわかります．さっそく，モニタ上で「D0790」と打って，先程のプログラムの中間言語を打出してみると，次のようになります．

```
☆0790番地
    0798  000A  B9  20  41  00
     ①     ②    ③   ④   ⑤   ⑥
☆0798番地
    07A4  0014  87  CD  20  FEF20014
     ①'    ②'   ③'  ③"  ④'     ※
    00
     ⑥'
☆07A4                ☆0035：36
    0000                  07A6
     ①"            (変数エリアスタート；テキスト終了)
```

BASICのプログラムは，行ごとにひとかたまりになっており，中間言語では，これら各行の区切りには必ず，NULL（$00）が置かれます．逆は必ずしも成り立たず，$00があるからといって，それが必ずしも行の区切りを表しているとは限りません．例えば，上の例では⑥と⑥'が行の区切りを表していますが，②，②'および※の中に含まれる$00は「行の区切り」ではないのです．この辺の事情は，後の章で説明する，行番号検索・非実行文のスキップなどで特に重要

な意味を持ってきます．

さて，各行の先頭には，次の行の先頭番地を指し示している「リンクポインタ」と呼ばれる２バイトのデータが置かれます．一見，無用の長物のようですが，ＧＯＴＯ・ＧＯＳＵＢ文の飛び先検索をはじめ，各種の検索操作の際に非常に役に立ちます．中間言語プログラムを見ていく際，このリンクポインタを手がかりにして辿っていけば，次の行がどこから始まるのかが一目瞭然であり，プログラムを見失わないで済む訳で，我々にとっても重宝なものです．実際，上の例でも①を見れば，次の行が＄０７９８番地から始まるとわかり，その手前の⑥の＄００が１行目の終りとなることがすぐにわかります．

次の，②や②′に書かれているデータは，それぞれの行の行番号を表す１６進数です．Ｆ－ＢＡＳＩＣでは，行番号として０～６３９９９の整数を使うことが許されていますから，各行ともこの位置には，＄００００から＄Ｆ９ＦＦまでの行番号定数が入ることになります．

３番目に位置する，③の＄Ｂ９，③′の＄８７や，③″の＄ＣＤなどは一体なんでしょうか．アスキーコードで＄Ｂ９は▼ケ▼というカナ文字のコードに該る訳ですが，実は，この＄Ｂ９こそが，▼ＰＲＩＮＴ▼を表現する中間コードなのです．同様に＄８７は▼ＧＯ▼を，また，＄ＣＤは▼ＴＯ▼を表現します．ＢＡＳＩＣインタプリタでは，プログラムを読み込んだ際に「予約語綴り」で始まる１節を発見すると，アスキーコードでカナ文字などの領域（＄８０～＄ＦＦ）に位置するコード，即ち，中間コードへと翻訳していくのです．ＢＡＳＩＣ内で，カナ文字の変数名・関数名が使えない理由もここにあるのです．従って，ＢＡＳＩＣプログラム中で，カナ文字を打ち込むと，すぐに「Syntax Error」が出てしまいます（もちろん，文字定数内のカナ文字や，注釈文字列中のカナ文字は別ですが）．プログラムの書き込み時においてこの「Syntax Error」が出るのは，上述のケースの他には，行番号が０～６３９９９の間にない場合と，▼ＧＯ▼のあとに▼ＴＯ▼もしくは▼ＳＵＢ▼がない場合（空白▼␣▼が間にいくら狭まっていても，後に▼ＴＯ▼または▼ＳＵＢ▼がありさえすれば，エラーは出ません）の２つだけなのです．ＢＡＳＩＣプログラムの地の文にカナ文字を打ち込むことが，いかに致命的なミスであるかがわかるでしょう．なお，どの中間コードがどのキーワード（予約語）に対応するかについては，６－４節「中間コード一覧表」などを随時参照して下さい．これらの表には，コードと予約語の対応だけでなく，各処理系のエントリ・アドレス（入口番地）も掲載してあります．

④および④′の＄２０は，「空白」を表すアスキーコードです．ＢＡＳＩＣプログラム中では，ひとつながりの言葉（予約語綴り・変数名・関数名など）の間に割

込まないかぎり，空白は無視されるようになっています．ですから，プログラムリストを出した際に見やすくなるように，空白を挿入・配置するのも，プログラミング上達のための一つのコツとなります．⑤の＄４１は，変数名「A」を表すアスキーコードです．変数などの取扱いについては，後の章で詳しく説明いたします．

さて，※の＄FEF20014という長い綴りは一体なんでしょうか．実は，これが「GOTO　20」の「20」を示す定数表現なのです．簡単に説明すると，最初の＄FEが「定数」であることを表す中間コードで，特に，**＄FEF2**で始まった場合，その直後の２バイトが「行番号」を表す定数値であることを示します．

最後に，①′の**＄0000**ですが，これはBASICのプログラムがここで終わりとなることを示す**区切文字**です．通常，リンクポインタが入るべき場所ですが，ここに，次の行の始まる番地の代わりに＄0000が入っていることを発見すると，インタプリタは，「プログラムはここで終り」と判断するわけです．（FM－7では，テキストエリアにかぎらず，リンクポインタ方式が好んで用いられていますが，この技法，すなわち「＄00ないし，＄0000が特殊な意味を表す」というやり方が必ず採用されているので，覚えておいて下さい）．

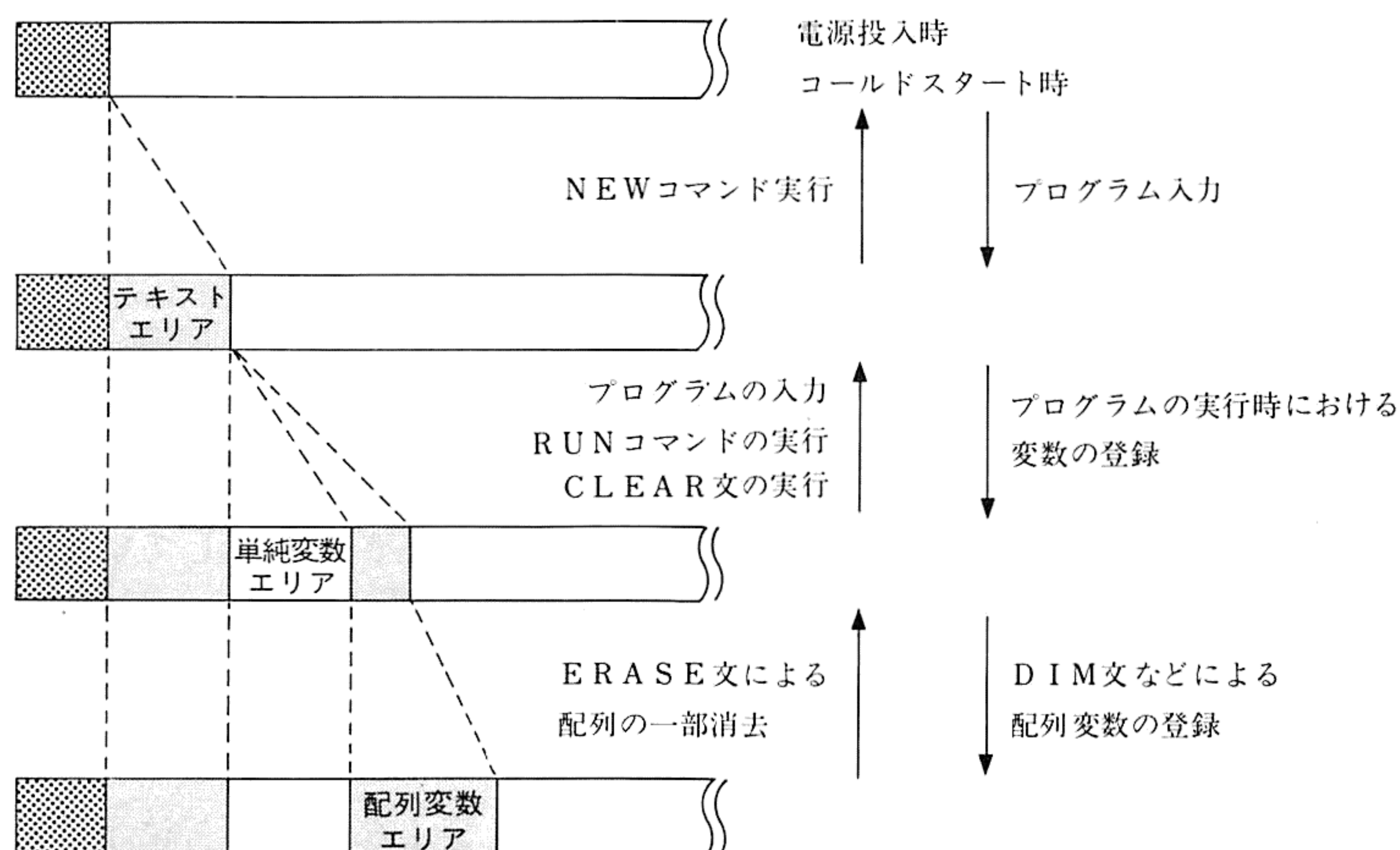
テキストエリアの後には変数エリアが来るわけですが，この変数エリアが始まる先頭番地を指すポインタ（35：36）は，確かに，プログラム終了の区切文字＄0000が書かれている次の番地（上の例では＄07A6番地）を指していることがわかります．

2—2　変数の格納形式（単純変数および配列変数）

単純変数エリア，ならびに配列変数エリアは，この順に，テキストエリアの後に置かれます．そして，それぞれのエリアの先頭番地を指しているポインタが，**図2・1・2**にも示したように，（35：36）および（3B：3C）です．テキストの後に動的に配置される関係上，電源投入時・リセット（コールドスタート）時・プログラム入力後・NEWコマンド実行後などでは，単純変数エリア・配列変数エリアともに，その大きさは**0**であり，（35：36），（3B：3C）両ポインタの値は，フリーエリア先頭番地（3D：3E）の値に等しくなります．テキストエリア・単純変数エリア・配列変数エリアの三者が，ダイナミックに変化していく様子を示したのが，**図2・2・1**です．

変数の登録・参照はすべて，＄950F番地から始まる，一連のサブルーチン群（我々は，このルーチンを「変数サーチ」と名付けました）によって行われます．この「変数サーチ」の節は第3章で扱うため，この節では，変数の格納形式などの説明をします．

図2・2・1　各エリアの伸縮の様子

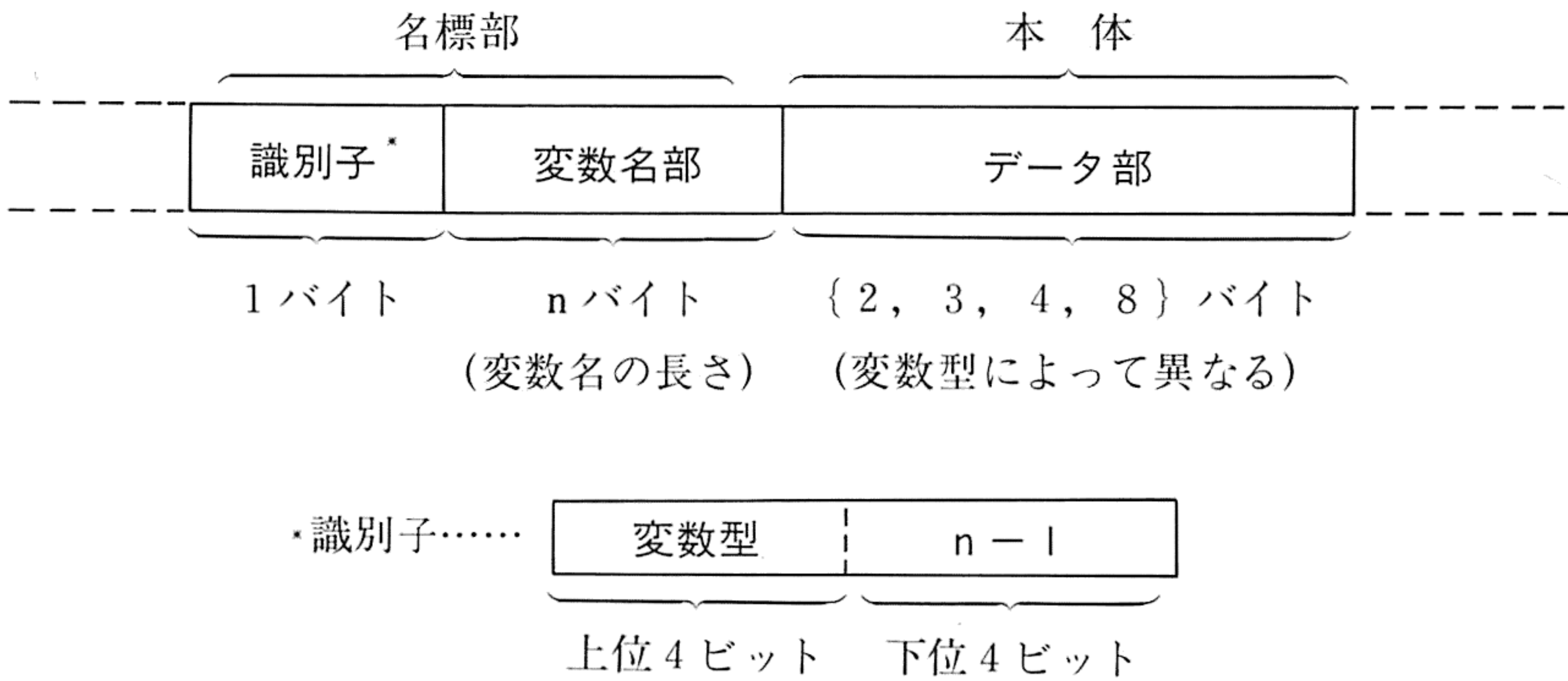
電源投入時
コールドスタート時
NEWコマンド実行
プログラム入力
テキスト
エリア
プログラムの入力
RUNコマンドの実行
CLEAR文の実行
プログラムの実行時における
変数の登録
単純変数
エリア
ERASE文による
配列の一部消去
DIM文などによる
配列変数の登録
配列変数
エリア

2—2—1　単純変数の格納形式

整数型・単精度ならびに倍精度実数型・文字列型の各単純変数は，ワークエリアのポインタ（３５：３６）が指す値を先頭番地とする単純変数エリアに，登録順いいかえれば**出てきた順で**格納されていきます．一つの単純変数を格納するのに要するメモリ容量は，変数名の長さと変数型の２つの要因によって決定されますが，この格納の形式を示したのが**図2・2・2**です．

図2・2・2

名標部		本　体
識別子*	変数名部	データ部
1バイト	nバイト（変数名の長さ）	{2，3，4，8} バイト（変数型によって異なる）

*識別子……

変数型	n－1
上位4ビット	下位4ビット

主な構造を説明します．

まず**識別子**ですが，**変数型を指示する上位４ビット**と，**変数名部の長さマイナス１**の値をとる１６進数が入る**下位４ビット**から成り立ちます．上位４ビットには，

整　数　型 ………＄２　　　倍精度実数型………＄８
単精度実数型………＄４　　　文 字 列 型 ………＄３

の各指標が格納されますが，この２，４，８，３の四つの指標は，ＦＡＣ１内のデータ型を表すワークレジスタ（＄１７番地）に格納される値でもあります．また，この指標の値は**同時に，各型の変数のデータ部の長さをも示しています**（実は，２，４，８，３の四つの指標は，変数のデータ部に要するバイト数に合わせて作られたものなのです）．

下位４ビットの方は，変数名部の長さが１～１６バイトですから※，それに応じて，＄０～＄Ｆの値が格納されます（※「Ｆ－ＢＡＳＩＣ文法書」に示してあるように，プログラムの側でいくら長い変数名を用いてあっても，実際に変数エリア内に登録されるのは頭から１６文字までです．この，変数名の中途切捨もすべ

て**変数サーチ**ルーチンで行われます).

このような識別子をわざわざ置いておく必然性については，以下に順を追って説明することにします.

(1) まず，**変数型指示子**（上位4ビット）が絶対に必要であるのはおわかりいただけるでしょう．各変数型が混ざり合ってエリア内に羅列される方式を採っている以上，この指示子がなれけば，後に続くデータが何のデータであるのか不明となり，変数情報が死んでしまうからです．しかし，各変数型に個別のエリアを設ける方式では，それだけポインタの数が増えてしまい，管理が複雑化するだけで実用的ではありません.

(2) 変数サーチ・登録を行う側にしてみれば,「できるだけ速やかに変数のピックアップや移動を済ませるような構造であってほしい」という要請があることです．これまでの説明からわかるように，変数エリアというのは不定長データの集合体です（サーチを行う側から見れば，エリア内に固定長データが並んでいた方が行いやすいわけですが，それでは，ほとんどのデータでＮＵＬＬ（＄００）や空白（＄２０）などの余分な文字を間に補うこととなり，メモリ効率が極端に悪化します)．そのため，不必要なデータはなるべく飛ばし読みする部分を多くする工夫が必要となり，識別子は飛ばし読みを行う上で２つの働きを持ちます.

(ｉ)捜している変数の識別子と比較することにより，変数型か変数名の長さが異なっている場合には，読み飛ばしを行い，次の場所を捜すようにします.

(ⅱ)識別子の上位４ビットは変数の**名標部の大きさ**を，下位４ビットは**データ部の長さ**をそれぞれ示し，両者の和プラス２という値を計算することによって，その変数がエリア内に占める大きさ（バイト数）を導くことができます．この値は同時に，読み飛ばしの幅でもあります.

このうち，（ｉ）の働きは，ルック・アヘッドという方式であり，例えば,「ＶＡＲＩＡＢＬＥ」および「ＶＡＲＩＡＢＬＥ２」という２つの変数が登録されている場合に，▼ＶＡＲＩＡＢＬＥ▼まで比較した後，次の文字▼２▼を見て初めて両者の識別ができるというのでは,非能率であり,毎回このような無駄をすることを避けるため，あらかじめ変数名の長さの比較で分類しています．また，(ⅱ)の方は，変数エリア内におけるリンクポインタの一種ということができます.

変数名部には，変数名を構成する文字列がアスキーコードで格納されます．識別子の項で述べたように，変数名部の大きさは１～１６バイトの範囲内となります．変数名の規則と照合すれば，この場所には英大文字か数字のコードしか入らないことになっていますが（「Ｆ－ＢＡＳＩＣ文法書」参照；なお，英小文字

はテキスト作成ルーチン内で大文字に変換されます），しばしばこの規則から外れた文字コードで始まる変数名が格納されることもあります．これはＤＥＦ␣ＦＮ文（ＦＮ関数定義文）中の関数名を単純変数エリア内に登録する際，一般の変数名と区別する目的で最初の一文字に＄８０を加えるという操作を行うことから起こる現象です．

最後に，データ部（変数本体）について説明します．次頁の**図２・２・３**に，変数名が▼ＡＢＣＤ▼であるとした場合の，各型の変数の格納形式を掲げます．

このうち整数型は，ＦＡＣの２バイトがそのまま格納されるだけで，特に問題ありません．

単精度実数・倍精度実数の各数値型の格納形式を御覧下さい．ＦＡＣでは符号部は独立した１バイトを占め，仮数部最上位バイト中のＭＳＢは常に１になっていますが，変数エリア上の数値は，メモリ節約のため**符号部が仮数部最上位バイト中のＭＳＢに圧縮された形で格納されており**，ＦＡＣにおける形式と異なっていますので注意を要します．このため，ＦＡＣ１のデータを変数型に直して格納するルーチン［＄Ｂ３３７番地から］，および，変数型のデータをＦＡＣ１にロードするルーチン［＄Ｂ３０１番地から］がインタプリタ内に用意されています．

文字列型の場合，数値型のように容易にはいきません．文字列型変数について理解するためには，インタプリタ内部で文字列がどの様に扱われているかについて触れておく必要があります．

まず，文字列データが他の数値データと異なる点に注目すると，数値データに関しては，整数型が２バイト，単精度型では４バイトという具合に長さが決まっている「固定長」であるのに対し，文字列データは長さが決まっていない，即ち「可変長」であるということが見えてきます．例えば，データが文字列▼Ａ▼であれば，データの長さは１バイトであるのに対し，データが▼ＡＢＣＤＥ▼であれば，長さは５バイトということになります．このように，データ長が変動することによって，文字列データの取扱いは非常に複雑なものとなり，様々な技法が必要になるわけです．

比較のため，**図２・２・４**の（１）を御覧下さい．図中の変数はすべて整数型とし，変数Ｂの内容を，現在の＄０４３Ｄから＄０３３Ｃに書換える必要が起きたとします．このとき，現在のデータ＄０４３Ｄと，書換用データ＄０３３Ｃの長さは共に２バイトですから，単に＄０３３Ｃを＄０４３Ｄのあった場所に置くだけで十分です．単精度型や倍精度型でも同様です．

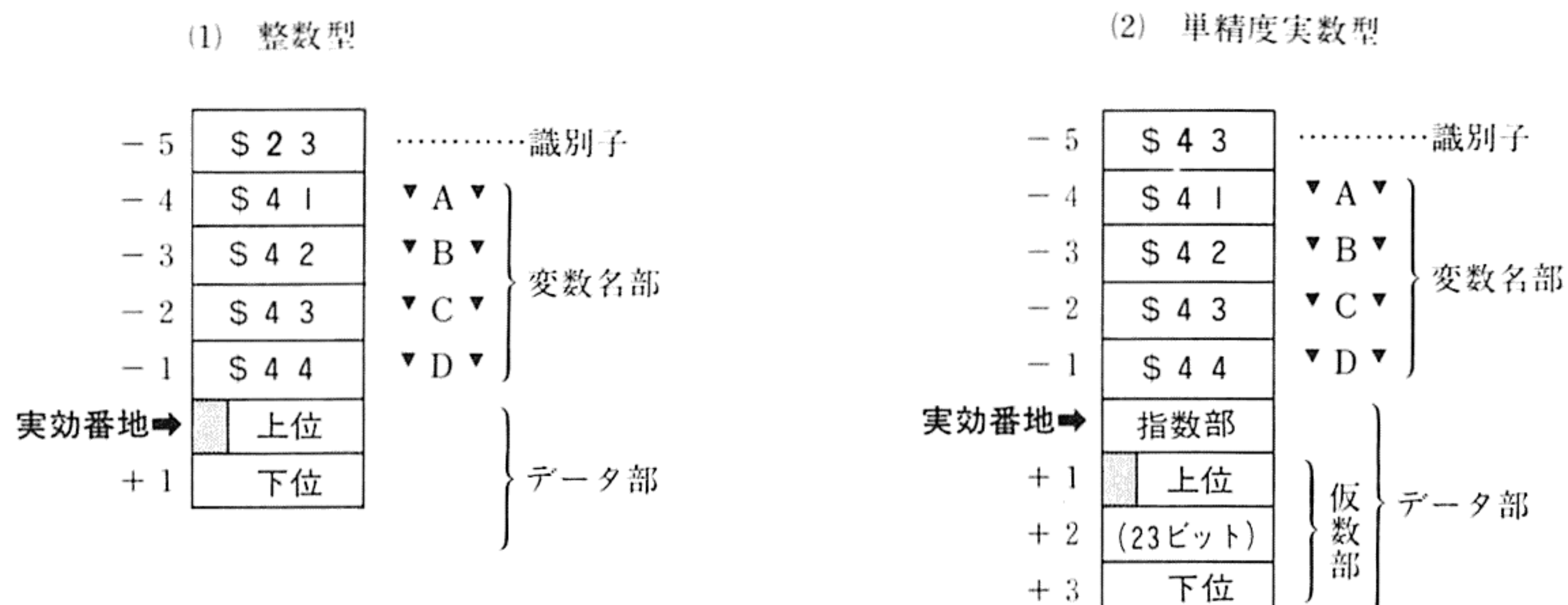
ところが，可変長データを扱う文字列型では，先程とは様子が異なってきます．既存データと更新データの長さが常に等しければ，数値型のデータと同じで，まった

図2・2・3　各型の単純変数の格納形式

(1)　整数型

番地	内容		
−5	$23	…………識別子	
−4	$41	▼A▼	変数名部
−3	$42	▼B▼	
−2	$43	▼C▼	
−1	$44	▼D▼	
実効番地➡	上位		データ部
+1	下位		

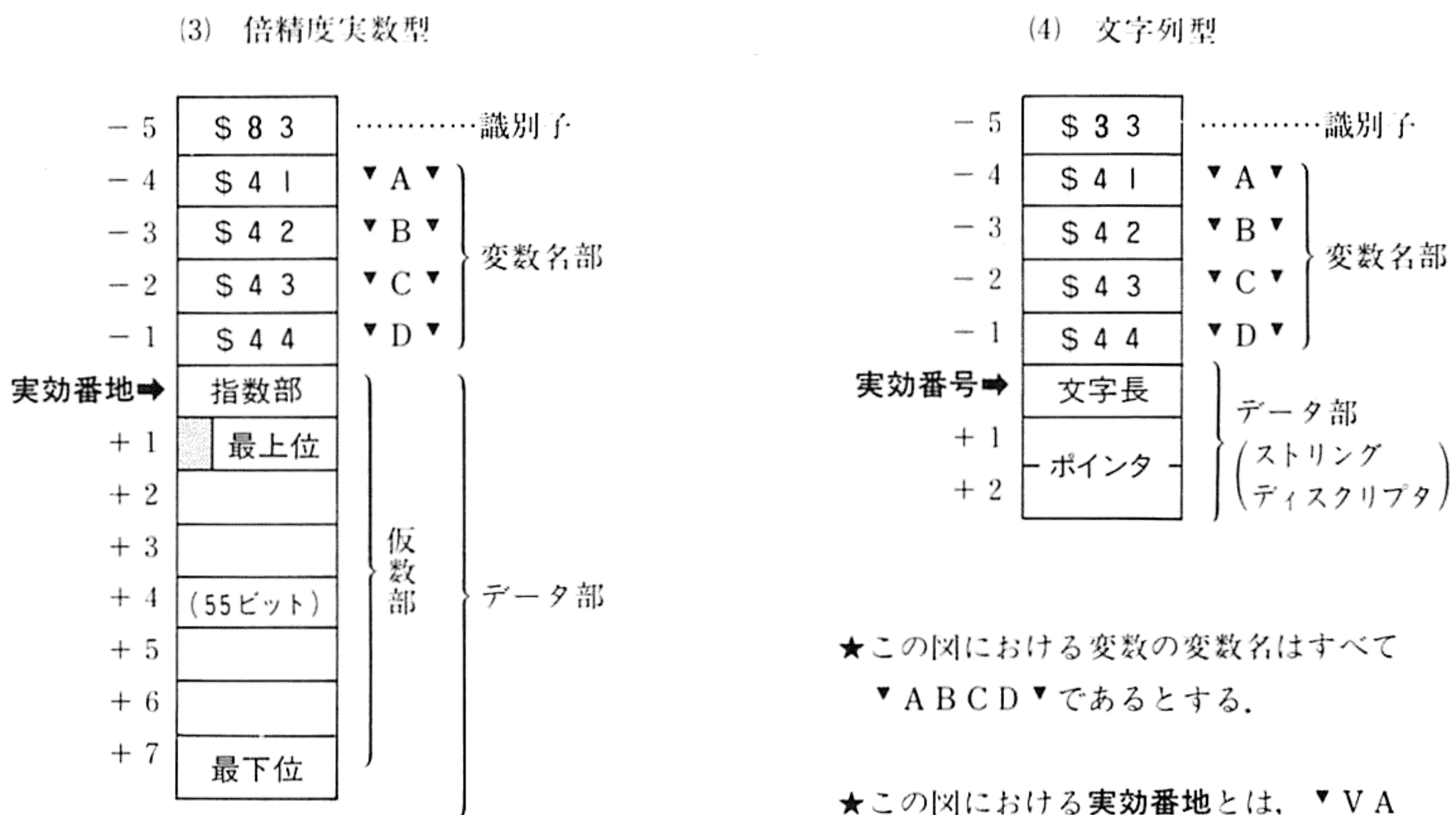
(2)　単精度実数型

番地	内容			
−5	$43	…………識別子		
−4	$41	▼A▼	変数名部	
−3	$42	▼B▼		
−2	$43	▼C▼		
−1	$44	▼D▼		
実効番地➡	指数部			データ部
+1	上位		仮数部	
+2	(23ビット)			
+3	下位			

★図中の□は，整数型・単精度実数型・倍精度実数型の仮数部のMSBであり，仮数部の符号が格納される．仮数部が正の数ならばこのビットは0（クリア）となり，負の数であれば，1（セット）となる．

(3)　倍精度実数型

番地	内容			
−5	$83	…………識別子		
−4	$41	▼A▼	変数名部	
−3	$42	▼B▼		
−2	$43	▼C▼		
−1	$44	▼D▼		
実効番地➡	指数部			データ部
+1	最上位	仮数部		
+2				
+3				
+4	(55ビット)			
+5				
+6				
+7	最下位			

(4)　文字列型

番地	内容		
−5	$33	…………識別子	
−4	$41	▼A▼	変数名部
−3	$42	▼B▼	
−2	$43	▼C▼	
−1	$44	▼D▼	
実効番号➡	文字長		データ部（ストリングディスクリプタ）
+1	ポインタ		
+2			

★この図における変数の変数名はすべて▼ABCD▼であるとする．

★この図における**実効番地**とは，▼VARPTR(ABCD)▼で示される番地のことを指す．

く問題は起こりませんが，せっかく用意された文字列型が常に同じ文字数のデータしか扱えないのでは面白くありません．可変長データを自由に取扱うことができる点こそ，インタプリタ型言語たるＢＡＳＩＣの特長の一つなのです（コンパイラ言語を使っている際，予め定義された文字数の文字列データしか扱えず，イライラすることもあります．配列宣言において，ＤＩＭ␣Ａ（２＊Ｘ＋Ｂ－５）などといったことが許されるというＢＡＳＩＣならではの融通性が，文字列処理においても要求されています）．

例えば，変数Ｂ内の既存データ▼Ａ▼に対し，更新データ▼ＡＢＣＤＥ▼を書き込もうとする際，更新データを前のデータ▼Ａ▼のあった場所にそのまま押し込めば，次のデータである変数Ｃが破壊されてしまいます．ですから，予め変数Ｃ以降のデータを後方に４バイト分ずらしてから，更新データ▼ＡＢＣＤＥ▼を書き込む必要があります（図（２）－ｉおよび（２）－ii参照）．同様に，更新データが既存データより短い場合でも，変数Ｃ以降のデータ群をあらかじめ前方にずらしておかないと，ＢとＣの間に空白地帯が生じ，無駄ができてしまいます．

前後にデータ全体をずらすこのような方法は，相当に手間のかかる作業を要します．そして，文字列変数を何十個，何百個と使用するプログラムにおいては，かなりの時間をこの操作だけで消費することになります．　例えば，ＢＡＳＩＣスクリーンエディタではこの方式が採用されておりますが（次の節のテキスト作成の項で説明します），巨大なプログラムの中に十行程度の修正を加えた場合，何秒あるいは何十秒もの間，画面表示が休止してしまうことも稀ではありません．エディタならば，それ程の高速性はなくとも十分ですが，頻繁に使用される文字列型変数においてはこのような低効率な作業ではとても間に合わないのです．

まず思い浮かぶ方法としては，文字列変数の長さを一律２５５文字なら２５５文字と定めておけば，手軽に能率を上げることができますが，この方法は識別子のところでも述べたように，メモリの使用効率が著しく低下してしまい，たかだかパソコンの記憶装置では，変数の登録数はいくらも増やせません．そこで，多くのパソコンでは，図（３）－ｉのように，文字列を実際に格納する領域（**文字列スタックエリア**）を別に設け，さらに変数名と実際のデータとを結合（リンク）するための**ポインタ**を作って，間接的にデータを参照するという方式 —— 間接表現法 —— が採用されている，というわけです．この方式ですと，変数名のあとには２バイトのポインタを入れておけばよく，データの書き換えは，スタックの最後尾に更新用の文字列データを付け加えてポインタをその最後尾を指し示す値に書き換えるだけで済んでしまいます．例えば，変数Ｂの内容を▼Ａ▼から▼ＡＢＣＤＥ▼に書き換える場合には，図（３）－iiのようにポインタを書き換えれば

図2・2・4　文字列処理の様子

(1). 整数型では

変数A
$00
0A
変数B
$04
3D
変数C
$FE
00

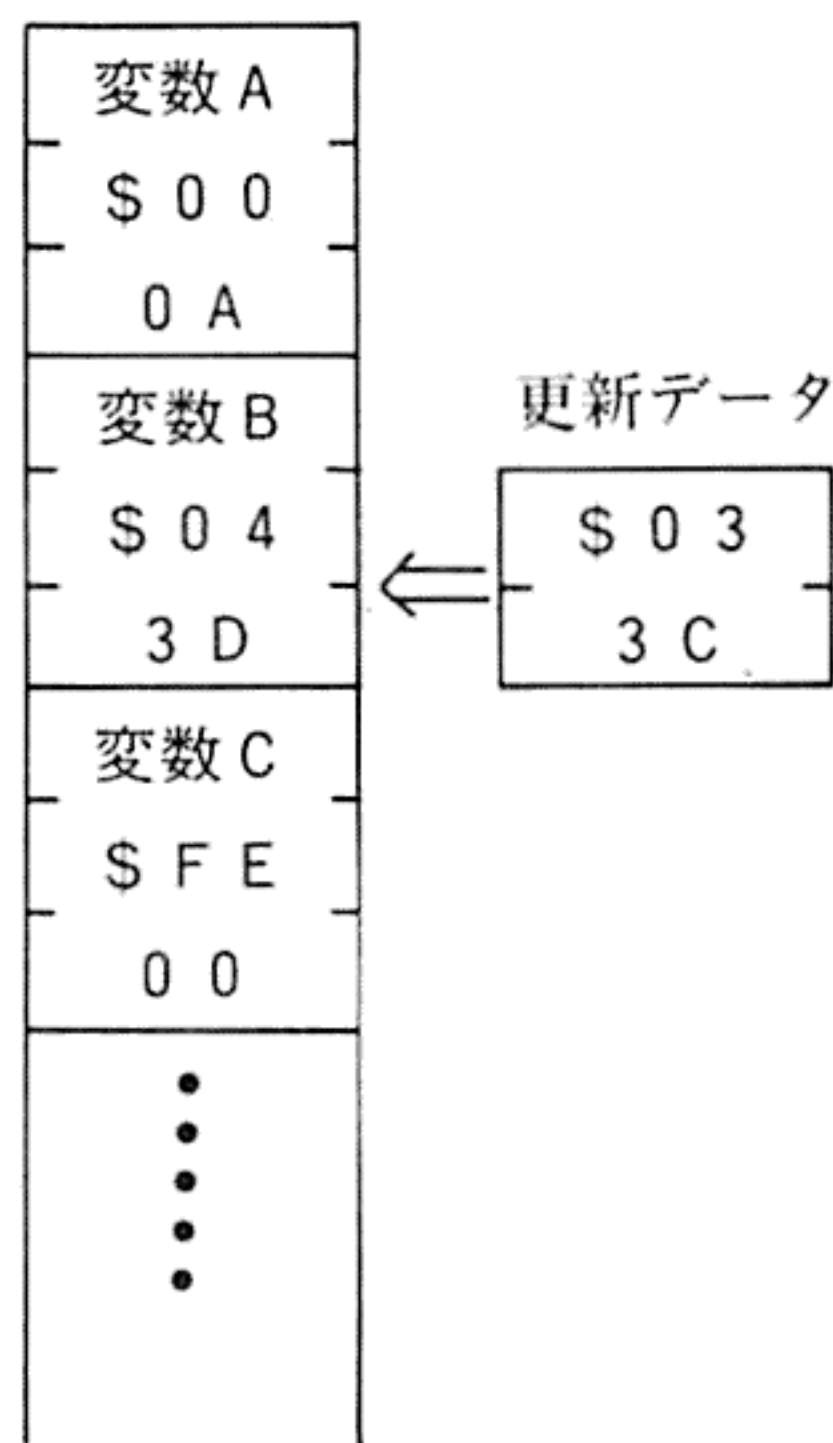
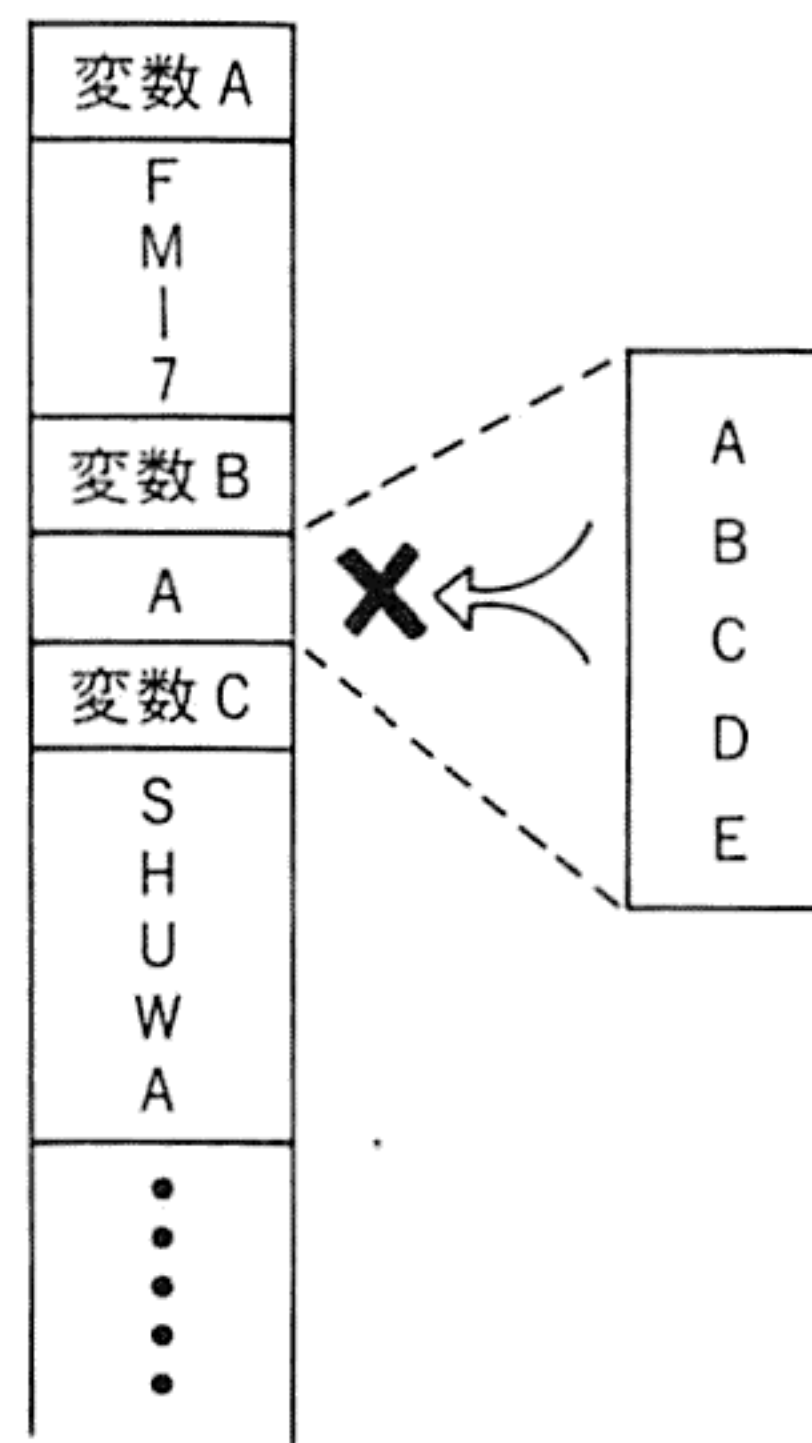
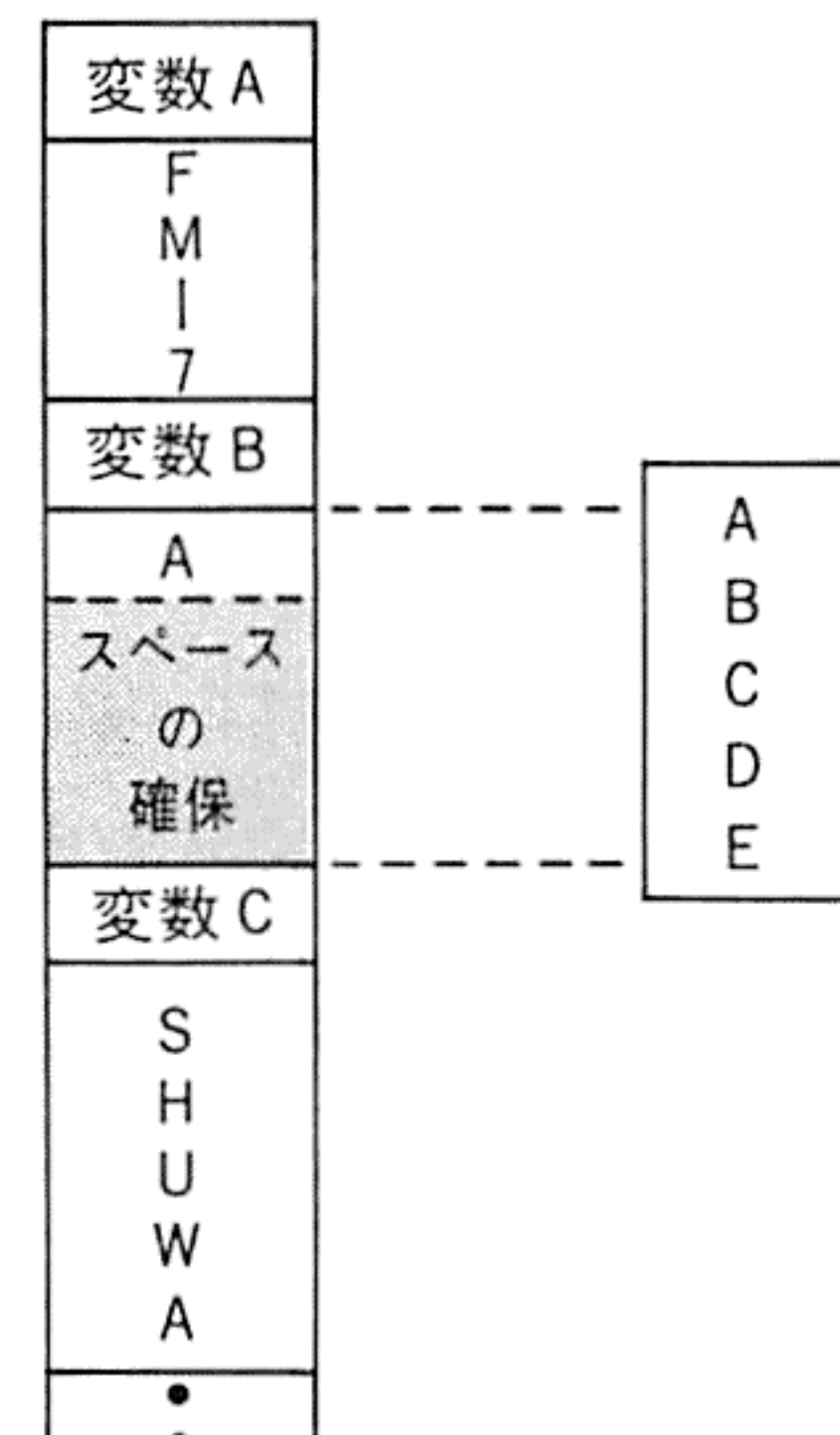
更新データ
$03
3C

(2)－ i . データの衝突

変数A
FM－7
変数B
A
変数C
SHUWA

ABCDE

(2)－ ii . データ移動による更新

変数A
FM－7
変数B
A
スペースの確保
変数C
SHUWA

ABCDE

※A＝"FM－7", B＝"A", C＝"SHUWA"の状態で, B＝"ABCDE"に更新しようとするには, あらかじめ, C以降を4バイトずつ後方にずらして, 格納スペースを作っておかなければならない.

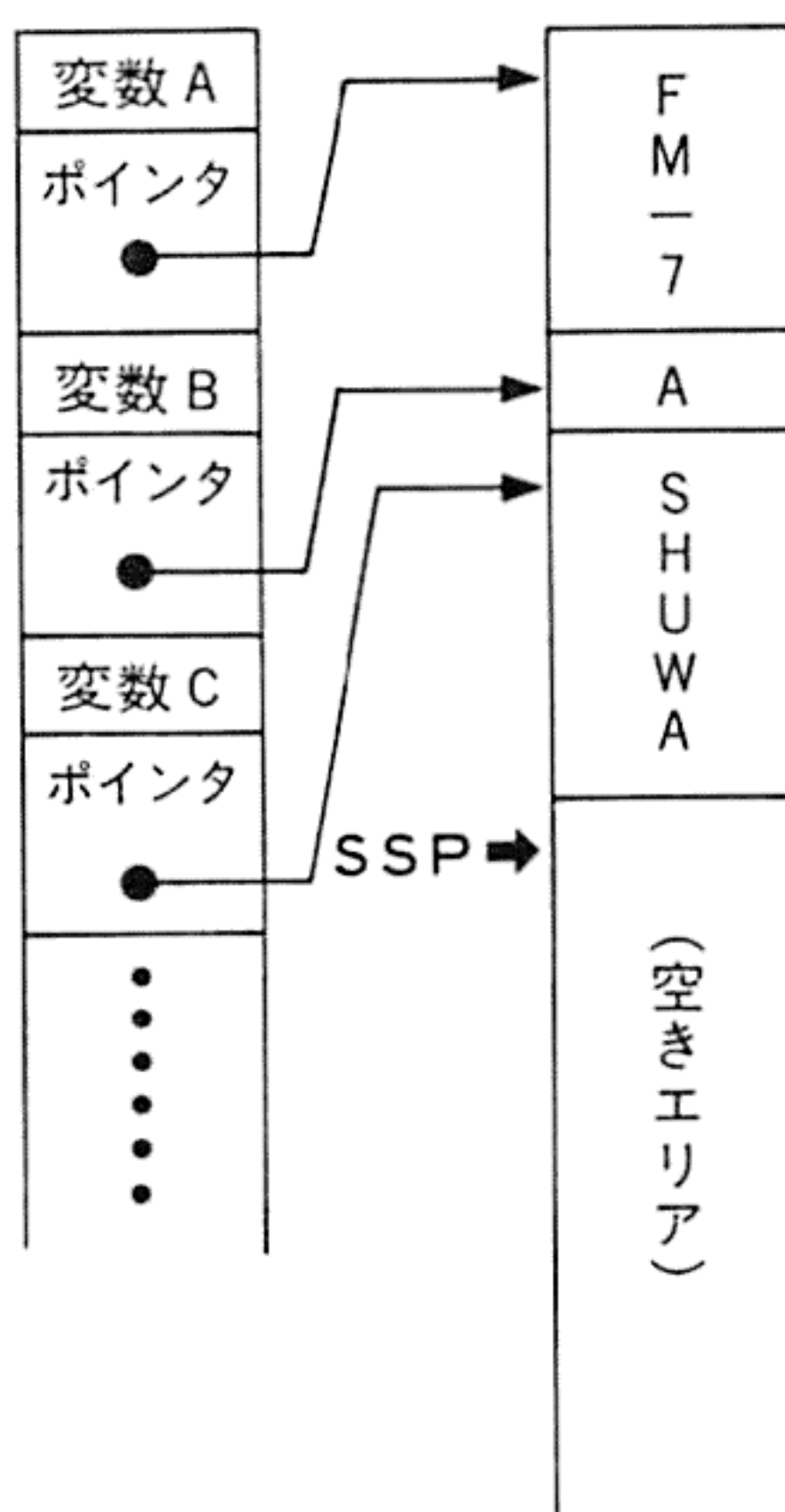
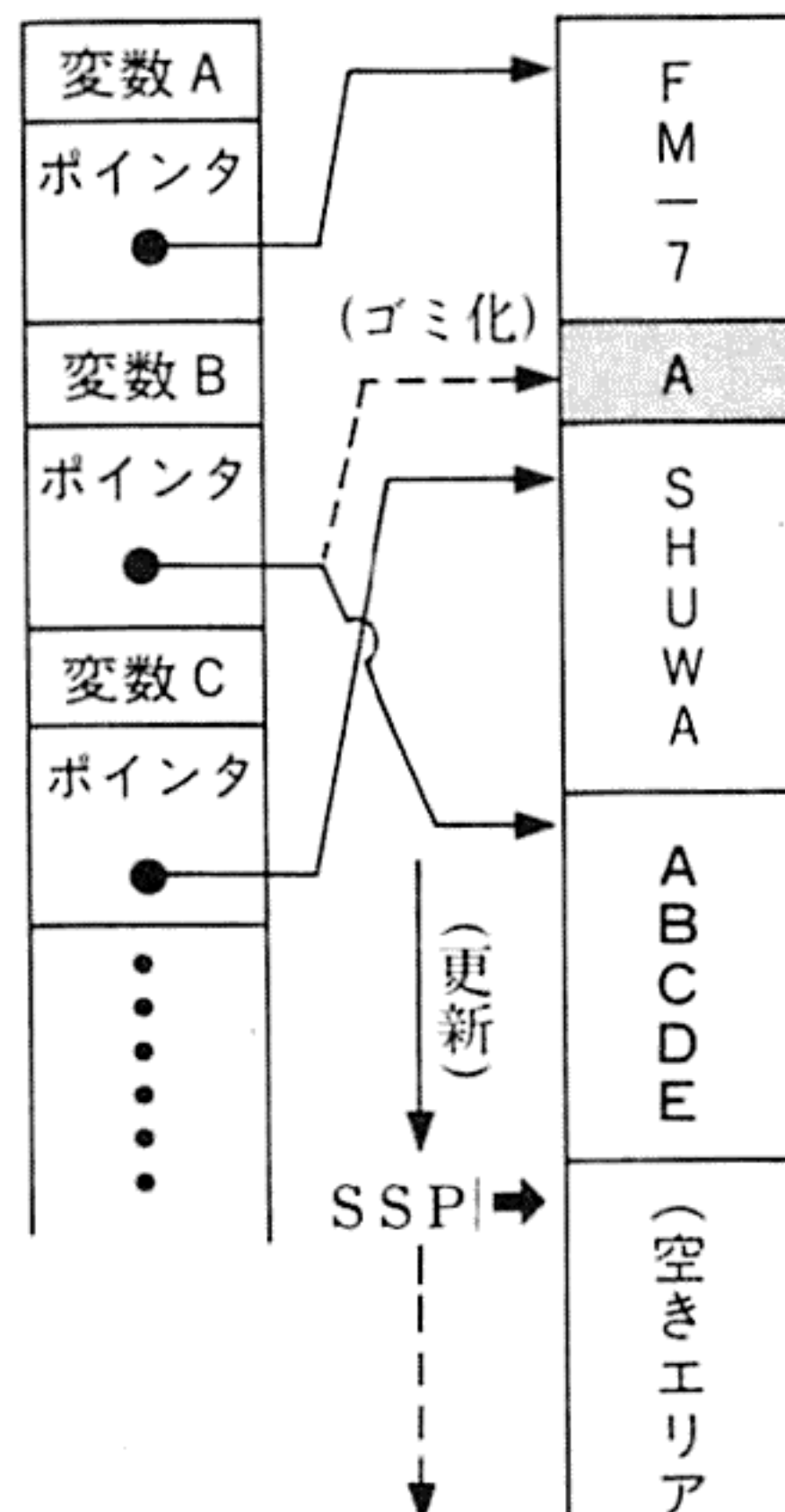
(3)－ i . 間接表現の採用

変数A ポインタ
変数B ポインタ
変数C ポインタ

FM－7
A
SHUWA
SSP
(空きエリア)

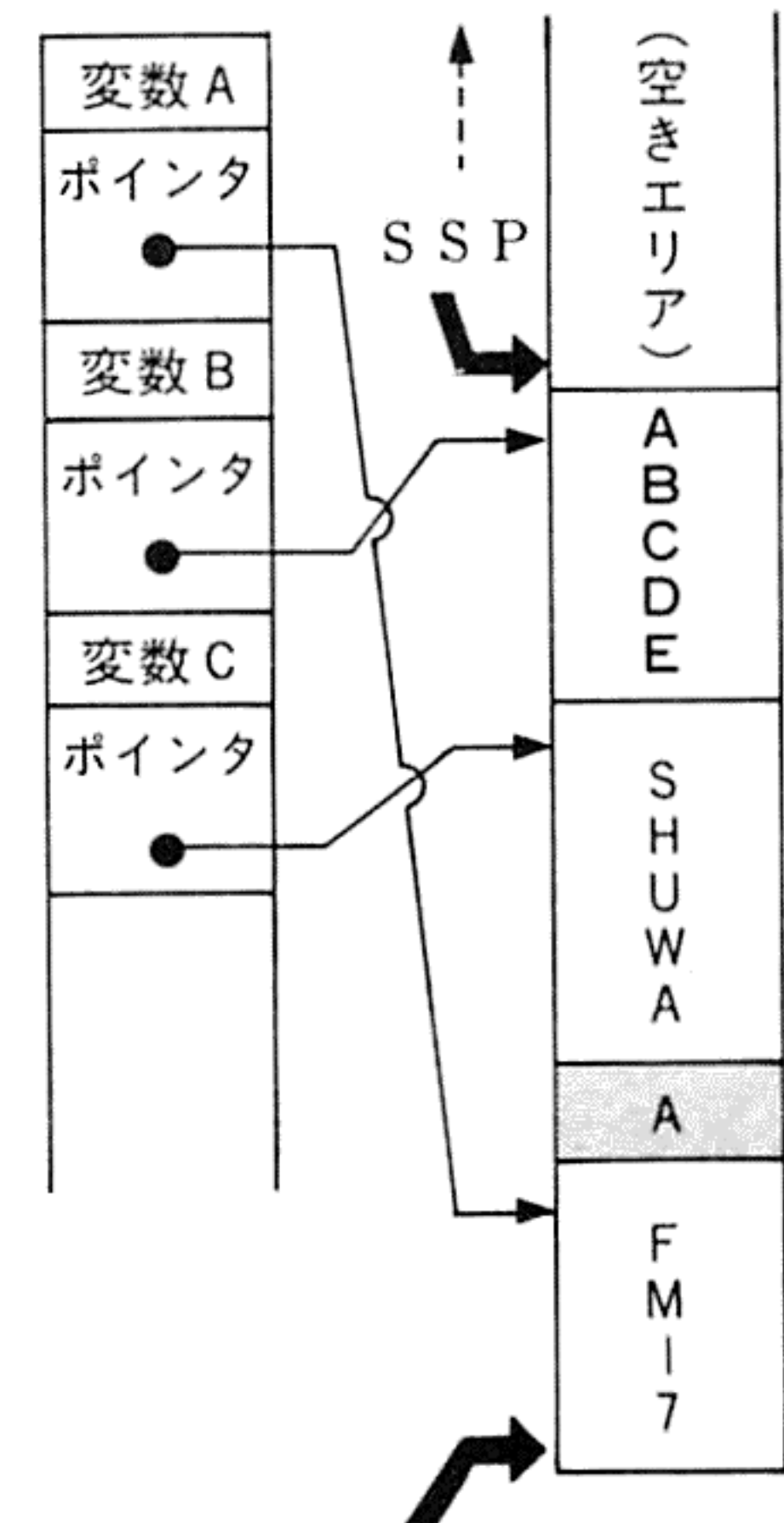
(3)－ ii . ポインタ書換えによる更新

変数A ポインタ
変数B ポインタ
変数C ポインタ

FM－7
(ゴミ化) A
SHUWA
ABCDE
(更新)
SSP
(空きエリア)

(3)－ iii . 実際のマシンでは

(上位アドレスの方向へ)

変数A ポインタ
変数B ポインタ
変数C ポインタ

SSP
(空きエリア)
ABCDE
SHUWA
A
FM－7

文字列スタックエリアの最下位アドレス

よいのです．ポインタならばデータ長は決まっていますし，文字列スタック上の古いデータ▼A▼は「**ゴミ**」としてそのままにしておけばよいので，その都度データをずらしたりする必要はなくなり，飛躍的に能率が向上します．

実際のマシンでは，他のエリアとの兼ね合いから，図(３)－iiiのように，下位のアドレスから上位のアドレスの方向へ，すなわち**ハードウェア・スタック**と同じ方向に文字列データが埋められていきます．また，使用中のエリアと未使用エリアとを分離するためのスタックポインタ（文字列スタックエリアのスタックポインタ，すなわちString Stack Pointer ですから，以後ＳＳＰと略します）が必要となり，ＦＭ－７では，ワークレジスタ（４１：４２）がＳＳＰの役目を果たしています．変数エリア側でも，ポインタだけではスタック上で自分のデータがどこまで続くか解らないので，ポインタに加えて**文字列の長さ**（文字数）を格納するレジスタが必要であり，両者を合わせた3バイトが**ストリングディスクリプタ**（以後，ＳＤと略すこともある）と呼ばれます．以上まとめると，文字列型変数の参照は，変数サーチ・ＳＤ読み込み・文字列スタック参照という３段階を経て行われるということになります．

ところで前述したように，パソコンの記憶容量には限界があり，文字列スタックエリアとして確保できるバイト数などは，わずかなものにすぎません．文字列の書き換えが頻繁に行われるようなプログラムにおいては，次々と更新が行われていく結果，すぐにエリアが埋まってしまいますが，この中には不要となった「ゴミ」の文字列が相当数含まれております．この際放っておいた「ゴミ」を一挙に掃除して片付けてやれば，再び，空きエリアを確保して使用可能な状態にすることができます．この「ゴミ掃除」のことを通常，**ガベージコレクション**(garbage collection —ゴミ集め—）と呼んでおり，文字列操作に必要な高度な技法（テクニック）の一つなのです．

このガベージコレクションには，**ゴミの選別**と**データの移動**，並びに**ＳＤ内のポインタの書換え**といった処理過程が必要であり，ガベージコレクションが始まるとそれだけで相当な時間を費してしまうことになります．それでもなお，図(２)の方式に比べると，格段に効率が良いのです．

以上の説明で，ＦＭ－７をはじめ各種のマイコンのインタプリタ内で，どのような形で文字列処理が行われるかについての，大体のイメージを摑んでいただけたと思います．なお，ＦＭ－７ではガベージコレクションの効率を上げるために，上記のシステムに加えて，さらに文字列スタックエリア中の各文字列がどのＳＤの指示対象となっているかを表現する，**逆向きのリンクポインタ**があります．すなわち文字列スタックエリア内における有効な文字列は，**図２・２・５**のように，

図2・2・5　文字列スタックエリア内のリンクポインタ

変数エリアなど　　文字列スタックエリア

変数エリアなど		文字列スタックエリア	
		S46(F)	←$abcd番地
$efgh番地→	S04	S4D(M)	文字列"FM−7"の実体部分
文字列"FM−7"を指すSD	Sab	S2D(−)	
	cd (ポインタ)	S37(7)	
		Sef (リンクポインタ)	
		gh	
		S41(A)	ゴミの文字列データ
		$00 (リンクポインタ(ゴミの大きさ))	←ゴミ識別子を兼ねる
		S01	
$mnop番地→	S05	S53(S)	←$ijkl番地
文字列"SHUWA"を指すSD	Sij	S48(H)	文字列"SHUWA"の実体部分
	kl (ポインタ)	S55(U)	
		S57(W)	
		S41(A)	
		Smn (リンクポインタ)	
		op	

文字列の大きさ

※文字列スタックエリア上の文字列データに関しては，SDのポインタとエリア上のリンクポインタとが，お互いの格納先頭番地を指し合っている．

SSP　　文字列スタックエリア最上位アドレ

文字列1	07 5F	文字列2	00 0A	文字列3	08 23	文字列4	08 61

$075F番地のSDで指される文字列　　文字長10バイトのゴミの文字列

SSP

文字列1	07 5F	文字列3	08 23	文字列4	08 61

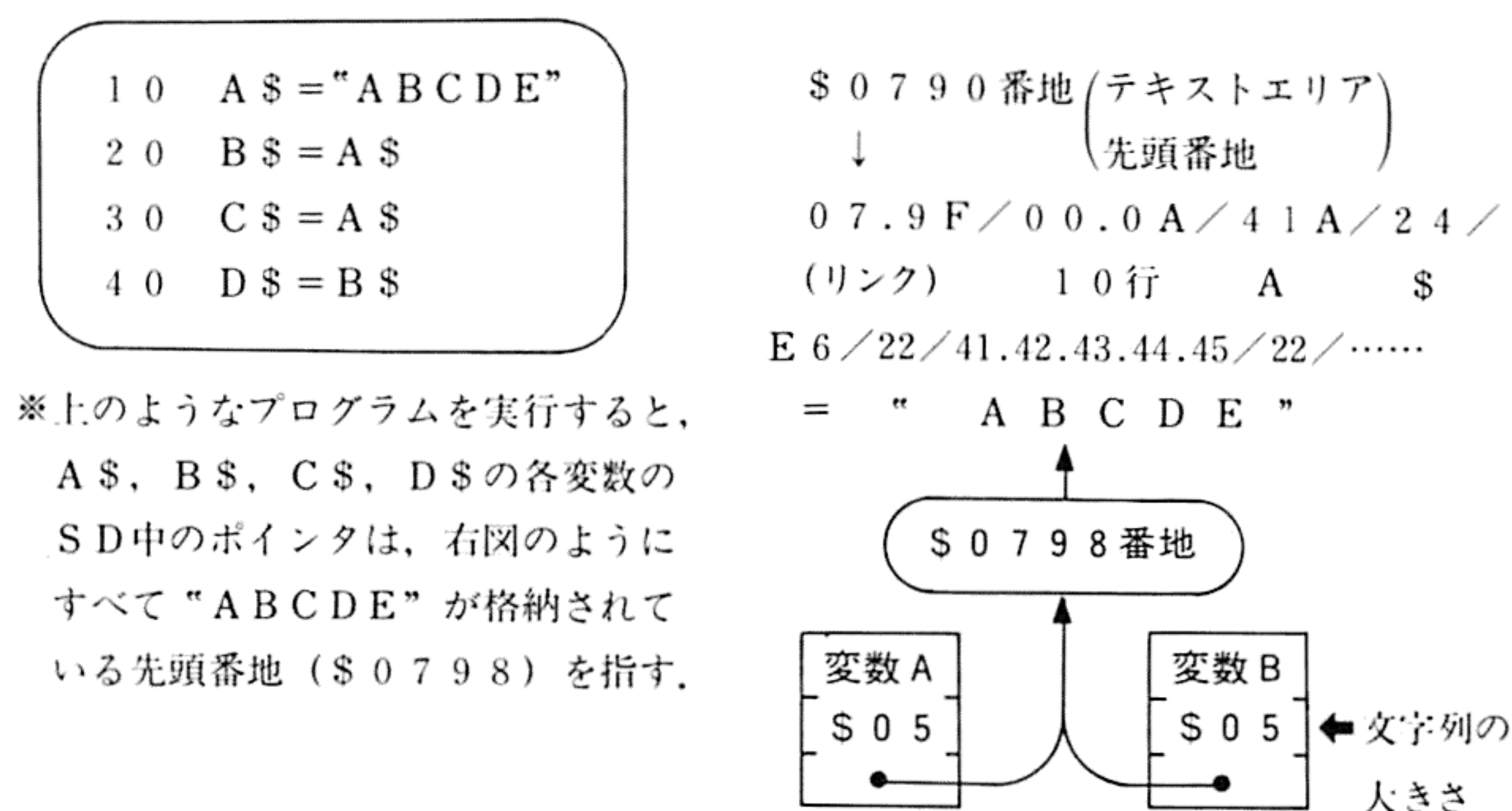
※ガベージコレクションの際には，ゴミである文字列2の分，すなわちリンクポインタ$000A（＝10）のバイト数プラス2だけ文字列1を移動して，文字列2をつぶせばよい．

ＳＤと相互に対応づけられているのです（文字列スタックエリア外のものは別です）．そして，ＳＤに対応づけられていない文字列こそが，有効でないデーター―すなわちゴミのデータであり，このとき，リンクポインタにはＳＤの格納番地の代わりに，**ゴミの大きさ**（バイト数）が入るのです．文字列の大きさは０～２５５文字の範囲――１６進数でいうと，＄００～＄ＦＦ――にあるので，リンクポインタの上位バイトは**＄００**となり，これが，**ゴミであることの識別子**をも兼ねているのです．このリンクポインタの導入によって，ゴミのデータを有効なデータから素速く見分けることができるようになり，しかも，データの移動に伴うＳＤの書き換えもリンクポインタのアドレスを読むことによりスムーズに行うことができるようになっています．このような方式は，インデックス修飾命令を豊富に備えた６８０９ＣＰＵの能力を最大限に引き出すものであり，かつ，６８０９ならではの特長であるといえます．

以上のような構造の下に，文字列型変数の場合，変数エリアの本体データ部には，３バイトのストリングディスクリプタが格納され，文字列本体は文字列スタックエリアその他に蓄えられるわけです．なお，「Ａ＄＝"ＡＢＣＤＥ"」などのように単純な文字列定数の代入文の場合，インタプリタは新たな文字列を作らず，テキストエリア（プログラム）中に書かれている文字列のアドレスをＳＤに書き込むだけで済ませてしまいます（**図２・２・６**参照）．前述したような，ガベージコレクションをはじめ，文字列スタックエリアに文字列の実体部分を格納することによって生じる負担を考えれば，インタプリタがこのような横着をする理由も納得できます．

図２・２・６　インタプリタの横着

```
１０　Ａ＄＝"ＡＢＣＤＥ"
２０　Ｂ＄＝Ａ＄
３０　Ｃ＄＝Ａ＄
４０　Ｄ＄＝Ｂ＄
```

※上のようなプログラムを実行すると，Ａ＄，Ｂ＄，Ｃ＄，Ｄ＄の各変数のＳＤ中のポインタは，右図のようにすべて"ＡＢＣＤＥ"が格納されている先頭番地（＄０７９８）を指す．

最後に，実例として

　　１０␣Ａ％＝１０：Ｂ！＝１２：Ｃ＃＝１４：Ｄ＄＝“ＡＢＣＤＥ”

といったプログラムを実行させた場合に，変数がどのように格納されていくかを見ることにします．格納されている数値の読み方は，符号部が圧縮されている点を除いて，ＦＡＣの数値の読み方とまったく同じです．ただし，標準ＲＯＭモード（ＤＩＳＫ，ＲＳ－２３２Ｃなどを使用しない）の場合を想定したため，テキストエリア先頭番地は，＄０７９０となります．それでは，**図２・２・７**を御覧下さい．

図２・２・７　実例による，変数の格納形式の様子

```
アドレス
0033 : 0790  07B6   (テキストエリア先頭番地，変数エリア開始番地)
  3B : 07CF  07CF   (配列エリア先頭番地，フリーエリア開始番地)

0790 : 07B4  000A   (テキストリンクポインタ，行番号［10］)          ┐
 794 : 41  25  E6  FE010A  3A      ┌☆▼FE01▼は        ┐           │テ
       (A   %   =   10［1バイト定数表記］:)  │1バイトの定数表記│           │キ
 79B : 42  21  E6  FE010C  3A      │であることを示す │           │ス
       (B   !   =     12     : )    └中間言語です.    ┘           │ト
 7A2 : 43  23  E6  FE010E  3A                                       │エ
       (C   #   =     14     : )                                    │リ
 7A9 : 44  24  E6  22 41.42.43.44.45 22 00                         │ア
       (D   $   =   "  A  B  C  D  E  " NULL(行の終り)              │
 7B4 : 00  00               ↑                                       │
       プログラムの終りを表す  $07AD番地                                ┘
       リンクポインタ            ↖
 7B6 : 20  41  00  0A            ＼_________________                 ┐
       (識別子▼A▼ 整数型の▼10▼)                      |                │単
 7BA : 40  42  84  40  00  00                          |                │純
       (識別子▼B▼  2³*0.1 100……  仮数部の符号はプラス)  |                │変
 7C0 : 80  43  84  60  00  00  00  00  00  00          |                │数
       (識別子▼C▼  2³*0.1  110……  仮数部の符号はプラス)  |                │エ
 7CA : 30  44  05  07  AD                              |                │リ
          ▼D▼                                         |                │ア
                     テキスト中の$07AD番地_____________|                ┘
            5文字の  (文字列▼ABCDE▼の先頭)
            文字列   を指すポインタ
            ストリングディスクリプタ(SD)
```

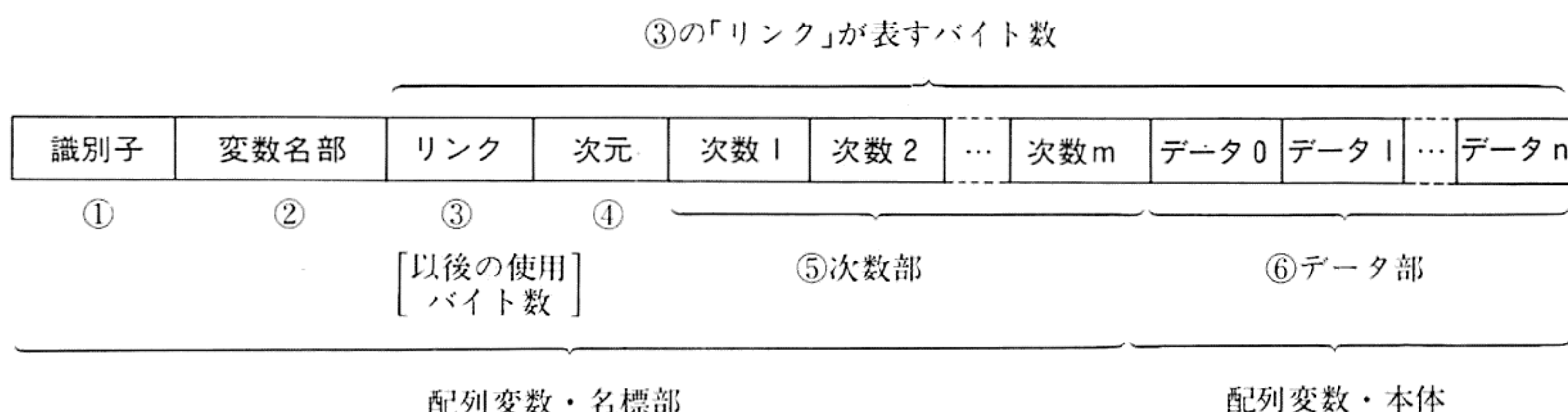
2－2－2　配列変数の格納形式

配列は，同じ型のデータが並んだ一種の構造体ですから，数値の格納には若干の工夫が必要になりますが，単純変数の格納形式がよく理解できていれば，難しいことはありません．図２・２・８に配列変数の格納形式を示します．

図２・２・８　配列変数の格納形式

③の「リンク」が表すバイト数

識別子	変数名部	リンク	次元	次数１	次数２	…	次数m	データ０	データ１	…	データn
①	②	③［以後の使用バイト数］	④	⑤次数部				⑥データ部			

配列変数・名標部　　配列変数・本体

それでは，以降に各部の説明をしていくことにします．

まず，①の識別子と②の変数各部については，単純変数とまったく同じです．

③の**リンク**の２バイトには，自分自身を含む，以後の使用バイト数が入ります．これは，変数サーチの際の，読み飛ばしのスピードを上げるためのものであり，変数名部（②）の次のアドレスに，この**リンク**の値を加えれば，**次の変数の先頭アドレス**が計算される訳です．

④には，配列の**次元**（２バイトで表現される）が格納されます．配列の次元というのは添字の個数であり，例えば，配列Ａ（ｎ）の次元は１，Ａ（ｍ，ｎ）では２，Ａ（ｉ，ｊ，ｈ，ｌ）では４となります．

⑤の次数部には，**各次元に対する次数**がそれぞれ格納されます．ここで，**次数**とは，**添字の最大値プラス１**のことであり，例えば，配列の次元が１であった場合には配列の全要素数に一致する値です．本書が，**次数**という紛らわしい表現をあえて使用するのは，配列の添字が０から始まるために⑤にはＤＩＭ文で宣言する添字最大値そのものではなく，添字最大値プラス１が格納されるため，**添字最大値**という表現が使用できないことに依ります．以後，本書で**次数**という場合，配列の添字最大値プラス１のことを示しますので注意して下さい．なお，配列の**全要素数**は，次数部に格納された**全次数の積**で計算されます．

⑥のデータ部は，配列の全要素の集合体です．整数型であれば２バイトずつ，倍精度実数型の場合には８バイトずつ区切られて格納されていきますし，文字列型であるなら，３バイトのＳＤ(ストリングディスクリプタ)が各要素の値となる

わけです．ここで注意しなければならないのは，**各要素が格納される順序**です．例えば▼ＤＩＭ␣Ａ（３，２）▼で宣言された配列であれば，

　　Ａ（０，０），Ａ（１，０），Ａ（２，０），Ａ（３，０），

　　　Ａ（０，１），Ａ（１，１），Ａ（２，１），Ａ（３，１）

の順序で格納されるのであって，決して，Ａ（０，０），Ａ（０，１）………という順序にはなりません．この点は，我々の日常感覚と異なっている部分なので，大いに気をつけて下さい．同じことは，⑤の次数部についてもいえます．

なお，①～⑤の部分を合わせて**名標部**，⑥のデータ部のことを**本体**と呼ぶことにします．

それでは，実際のプログラムで見ることにします．次のようなプログラムを実行させて下さい．

```
１０␣ＤＩＭ␣Ａ％（３，２）
２０␣ＦＯＲ␣Ｉ＝０␣ＴＯ␣３
３０␣␣ＦＯＲ␣Ｊ＝０␣ＴＯ␣２
４０␣␣␣Ａ％（Ｉ，Ｊ）＝Ｉ＊２５６＋Ｊ
５０␣␣ＮＥＸＴ
６０␣ＮＥＸＴ
```

この後，ＰＲＩＮＴ␣ＨＥＸ＄（ＶＡＲＰＴＲ（Ａ％（０，０）））やＭＯＮ／Ｄ００３Ｂなどの操作によって配列の位置を調べて，スクリーンにダンプさせてみると，配列変数の構造を観察することができます．その様子をわかりやすく描き直したのが，次頁の**図２・２・９**です．

ここまでの説明で，変数の構造について，およそのイメージを摑んでいただけたかと思います．さて，いよいよ，ＢＡＳＩＣ中間言語の作成と解読・実行について説明することにします．

図2・2・9　配列変数格納の実例（2の1）

☆テキストエリア

```
アドレス   リンクポインタ
0790： 07A2  000A  8420 4125 28 FE01032CFE0102 29 00
                   10␣  DIM␣  A %  (   3   ,   2   ) NULL
07A2： 07B4  0014  8120 49E6FE010020 CD 20 FE0103 00
                   20␣  FOR␣  I =  0  ␣ TO ␣  3
07B4： 07C7  001E  208120 4AE6FE010020CD20 FE0102 00
                   30␣  ␣FOR␣  J =  0  ␣TO␣  2
07C7： 07DE  0028  2020412528492C4A29E649DBFE020100D94A00
                   40␣  ␣ ␣ A % ( I , J ) = I * 256 + J
07DE： 07E5  0032  20 82 00
                   50␣  ␣ NEXT
07E5： 07EB  003C  82 00
                   60␣  NEXT
07EB： 0000
```

☆　ワークエリア

		プログラム実行前 ⇨ プログラム実行後	
0033：	0790（テキスト先頭）	33：	0790
0035：	07ED（単純変数エリア）	35：	07ED
003B：	07ED（配列変数エリア）	3B：	07F9
003D：	07ED（フリーエリアスタート）	3D：	081A

☆　単純変数エリア（実行後）

07ED ： 40　49　83000000……………単精度型変数 ▼I▼（値は4になっている）
07F3 ： 40　4A　82400000……………　〃　▼J▼（〃 3　〃　）

☆　配列変数エリア（実行後）

DIM A%（3 , 2）

07F9：	20	41	001F	02	0003 ¦ 0004	0000 ¦ 0100 ¦ 0200 ¦ ……… ¦ 0302
	識別子	変数名	リンク	次元	次数部	A%(0,0) A%(1,0) A%(2,0) …… A%(3,2)
						データ部（本体）

VARPTR（A%（0, 0））で示される番地
［この例では，$0802番地となる］

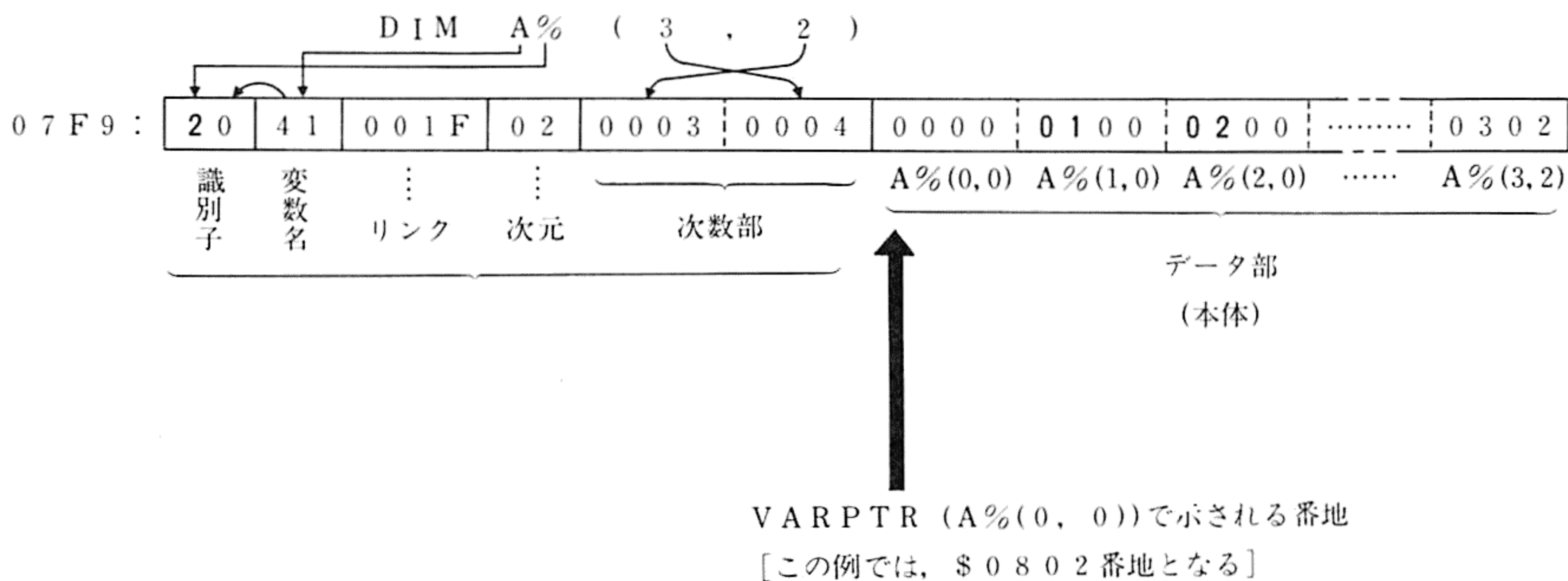



図2・2・9　配列変数格納の実例（2の2）

☆配列変数エリア（実行後）

VARPTR( )を基準とする相対アドレス	[今回の例における絶対アドレス]		内容		
−9	[$07F9]	→	20……………識別子 [整数型変数で，変数名の長さが(0＋1)バイト]		名標部
−8	[$07FA]	→	41……………変数名 ▼A▼		名標部
−7	[$07FB]	→	001F………リンク [以後の使用バイト数が31であることを示す]		名標部
−5	[$07FD]	→	02……………次元		名標部
−4	[$07FE]	→	0003………後の方の添字▼2▼に対する次数	次数部	名標部
−2	[$0800]	→	0004………前の方の添字▼3▼に対する次数	次数部	名標部
VARPTR(A%(0,0))で示される番地	[$0802]	→	0000……　A%（0，0）の値	A%(＊，0)	データ部（変数本体）
＋2	[$0804]	→	0100………A%（1，0）の値	A%(＊，0)	データ部（変数本体）
＋4	[$0806]	→	0200………A%（2，0）〃	A%(＊，0)	データ部（変数本体）
＋6	[$0808]	→	0300………A%（3，0）〃	A%(＊，0)	データ部（変数本体）
＋8	[$080A]	→	0001………A%（0，1）〃	A%(＊，1)	データ部（変数本体）
＋10	[$080C]	→	0101………A%（1，1）〃	A%(＊，1)	データ部（変数本体）
⋮	⋮		⋮		
＋20	[$0816]	→	0202………A%（2，2）の値	A%(＊，2)	データ部（変数本体）
＋22	[$0818]	→	0302………A%（3，2）〃	A%(＊，2)	データ部（変数本体）

2－3　ＢＡＳＩＣメインルーチン

——ＢＡＳＩＣテキストの作成と解読・実行——

2－3－1　ＢＡＳＩＣメインルーチンの構造

ＢＡＳＩＣメインルーチンは，次の３つの部分に大きく分かれます．

プロンプト表示部	（＄８Ｅ６３～＄８Ｅ８７）
テキスト作成部——テキスト作成ルーチン	（＄８Ｅ８８～＄８Ｆ１Ｂ）
解読・実行部———解読・実行ルーチン	（＄Ｃ６３Ｄ～＄Ｃ６ＥＣ）

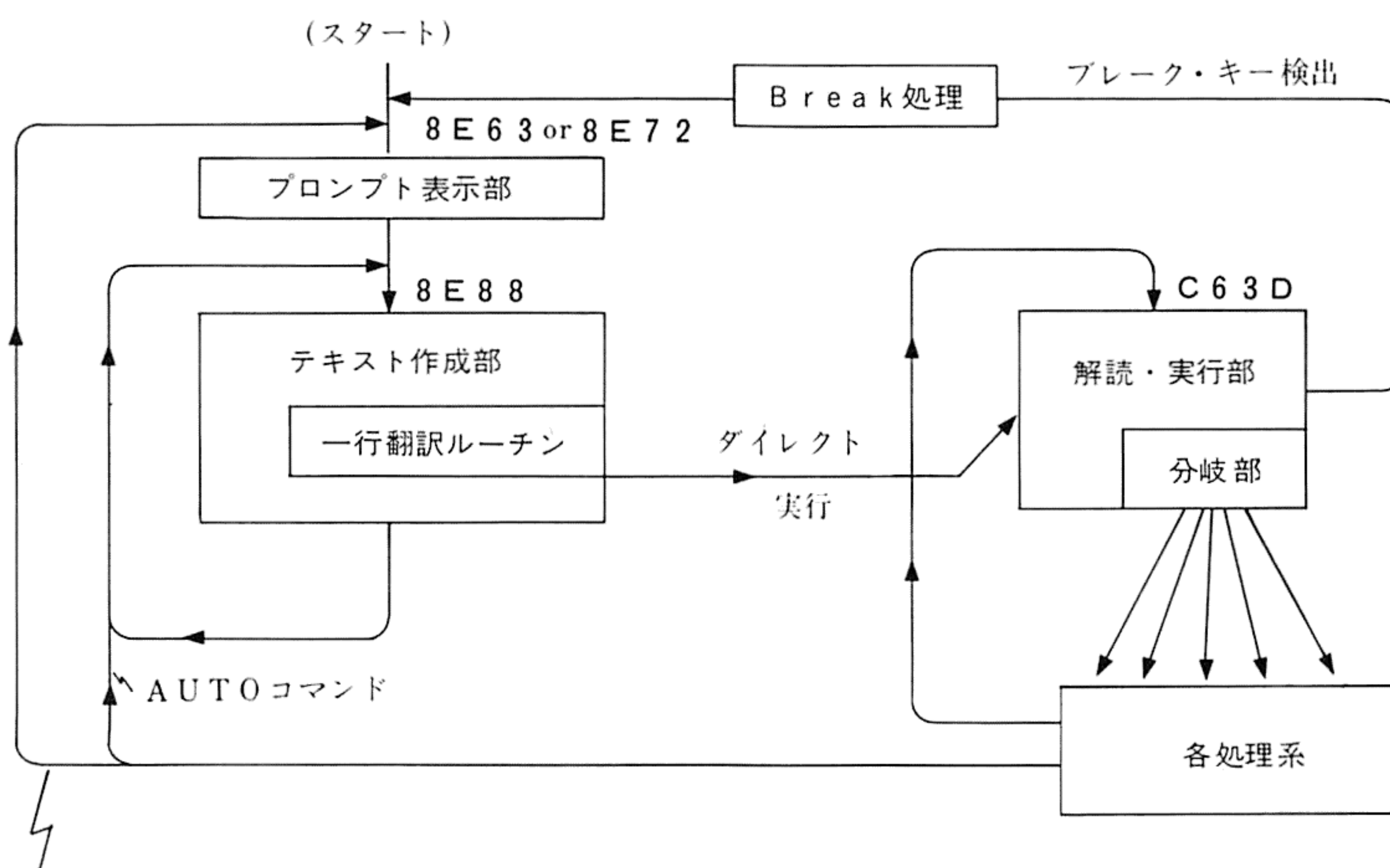
このうち，**プロンプト表示部**は，コールドスタート，ホットスタートなどのメインルーチン・エントリであり，コールドスタート時・ホットスタート時・ブレークキー入力時・ＥＮＤ文実行などプログラム実行終了時・ＳＴＯＰ文実行時・エラー発生時など，すべてプロンプト表示部をとおしてメインルーチンに入ります．コールドスタート時やプログラム終了時などでは，＄８Ｅ７２番地がエントリ・ポイントとなり，「Ready」出力のみを行うのに対し，ホットスタートやプログラム中断時（ブレークやエラー発生時ほか）では，＄８Ｅ６３番地がエントリ・ポイントとなり，「Ready」出力に先立って，「Break」「Abort」「Error」「Break␣In␣（行番号）」「Abort␣In␣（行番号）」「Error␣In␣（行番号）」のいずれかが，中断時の状況に応じて表示されます．

テキスト作成部は，プロンプト表示部と一つながりになっているのですが，一応，一つの独立したルーチンとして扱います．テキスト作成ルーチンは，独立した形の＄Ｃ２８８番地以下の**一行翻訳ルーチン**を，サブルーチンの形で含んでおりますが，一行翻訳ルーチンが単独で使われるのは特殊な例を除いて極めて稀ですので，両者を別々のものとしては扱いません．なお，このテキスト作成ルーチンは，キーボード入力はもちろんのこと，ＬＯＡＤ・ＭＥＲＧＥ・ＣＨＡＩＮなどによる，アスキーセーブされたプログラムの読込みにも用いられます．

解読・実行部は，各コマンドの処理系へ分岐する部分をサブルーチンの形で内包していますが，このことは言い換えれば，各コマンドの処理系はすべて解読・実行ルーチンのサブルーチンの一部となっている，ということです．従って，

各コマンド処理ルーチンの最後は，スタック内にネスティング用データを残して終了するFOR～NEXT文・WHILE～WEND文・GOSUB文を除けば，RTS（リターン）命令で終わっています（見かけ上はRTS命令になっていないものであっても，RTS命令を含む部分に分岐するなど，実質上のRTS命令を含んでいることになります．もちろん，エラーが発生した場合や，以下に述べるようなダイレクト実行用のコマンドやプログラム実行を停止させるコマンドの処理ルーチンについてはこのかぎりではありません．

メインルーチンの3つの部分の関係を図示したのが，**図2・3・1**です．プロンプト表示部からは，必ず，そのままテキスト作成ルーチンへ入ります．テキスト作成ルーチンが一行翻訳によってテキストバッファまで作成する段階で，行の先頭に行番号がなければ，ダイレクト実行文とみなして翻訳ののち解読・実行ルーチンに分岐し，行番号付きなら，テキスト中の格納位置を計算してその行を挿入したのち，再び，テキスト作成ルーチンの先頭に戻ります．解読・実行ルーチンは，各処理系へ分岐する基地となっていて，中間言語を解読して分岐アドレスを算出したあと各処理系ルーチンに制御を渡します．それぞれのコマンドの処理が終了すると，大抵の場合は再び解読・実行ルーチンに戻ってきますが，END文やSTOP文を実行したり，LIST・SAVE・DELELEなど主として直接モードで用いる――言い換えれば，ダイレクト実行用の――コマンドを実行した等の場合には，所定の手続きを終えたのち，プロンプト表示部に入ります．

図2・3・1　メインルーチンの3つの顔

2－3－2　メインルーチンにおける重要なワークエリアなど

まず初めに，ワークエリア内の**ポインタ（D9：DA）**と，このポインタを使用する**汎用読込みルーチン「D2」**と**「D8」**および，「D2」「D8」の後処理サブルーチン**「C760」**（数字・区切文字検出ルーチン）は重要なもので，第0章の(6)のこれらに関する説明を随時御参照下さい．

ワークレジスタ**（AF：B0）**は，**SP**（スタックポインタ）の復帰用の値を保存しておく待避場所で，文字領域上限（3F：40）マイナス1が初期値として入り，解読・実行ルーチンを通るごとに**その時点でのSPの値**に更新されていきます．このレジスタによって，例えば，エラーが発生した場合でも，SPの値をエラー前の値に戻してエラー前の情報を得ることができるわけです．

また，ワークレジスタ**（00BF）**も重要なものです．（BF）には現在扱っているファイルのファイル番号が格納されますが，普段は，＄00が入っており，コンソールが対象ファイルであることを示しています．LOAD，SAVEなどや，PRINT＃文，INPUT＃文など，特別なファイルをアクセスする際に該当ファイル用のファイルバッファの番号が格納されることになります．特に，テキスト作成ルーチン内で（BF）が＄00以外の有効な値を持っている場合は，キーボード入力によるテキスト作成ではなく，LOAD，MERGE，CHAINなどによって外部ファイルからプログラムを読み込んで，テキストを作成中である，ということを表します．従って，(BF)≠0で，かつ，行番号の付いていない行が入力された場合には「Direct In File」エラーが出力されます．

（33：34），（35：36），（3B：3C），（3D：3E）は，前述したように，それぞれテキストエリア，単純変数エリア，配列変数エリア，フリーエリアの各先頭番地を表すポインタです．ここで注意しておきたいのは，**（33：34）マイナス1番地，**すなわち，テキストエリア直前の1バイトには常に＄00の値を入れて開けておく必要がある，ということです．解読・実行ルーチンは，入口で，ステートメントの**区切文字**（▼NULL▼または▼：▼）のチェックをするため，この番地の内容が＄00（または＄3A）でないと，Syntax Errorを発生させてしまい，いくらRUNさせても，一向にプログラムが走りません．

さらに，（47：48）と（4D：4E），および（49：4A）と（4F

：５０）について触れておくことにします．

（４７：４８）には，**現在実行中の行番号**が格納され，これもまた，解読・実行ルーチンで更新されます．（４７：４８）の値はエラー処理ルーチン内でエラー発生行番号として参照されるほか，ＧＯＳＵＢ文などでは，リターン行番号としてスタックにプッシュされます．なお，コマンド待ちやダイレクト実行中では，（４７：４８）には＄ＦＦＦＦ（この値は行番号としては許されていないので混乱することはない）が格納され，「Illegal Direct」のチェックなどに用いられます．（４Ｄ：４Ｅ）は上のものとペアになっていて，**現在実行中のテキストの番地**が格納されるポインタです．これも，ＧＯＳＵＢ文などでスタックにプッシュされます．これに関連して，（４９：４Ａ）および（４Ｆ：５０）では，それぞれ，**ＣＯＮＴ**コマンドでプログラムを継続実行する際の**再スタート行番号**，並びに，**再スタート用テキスト番地**が格納されます．なおこの（４９：４Ａ）は，テキスト作成中においては，これから書込みを行うべきテキスト番地の一時退避用として使われます．

テキストエリアをいじっていたら……なんと，ＦＭ－７が走り出した!?

最後に，**対サブルーチン情報受渡用レジスタ（４Ｂ：４Ｃ）**について説明します．これは要するに，AccDの延長に相当するものです．６８０９ＣＰＵには４本のポインタを除くとレジスタはAccDしか残りません．これ一つではサブルーチンとの情報のやりとりに不便なため，上記のレジスタをワークエリア内に設けたわけです．主に行番号関係のルーチンでよく使われますが，AccDやＸレジスタなどの退避用としても多用され，きわめて汎用性の高いレジスタなのです．

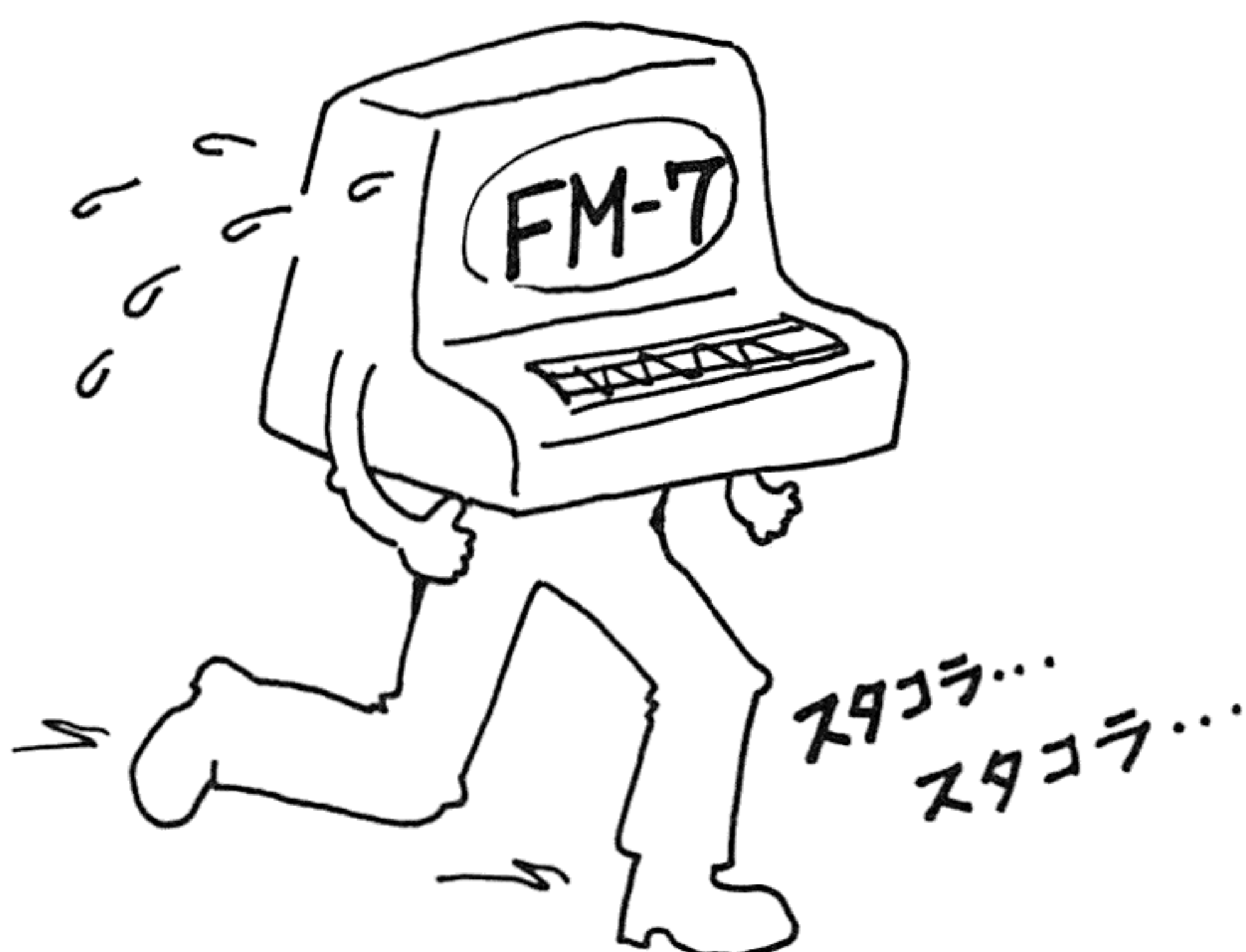
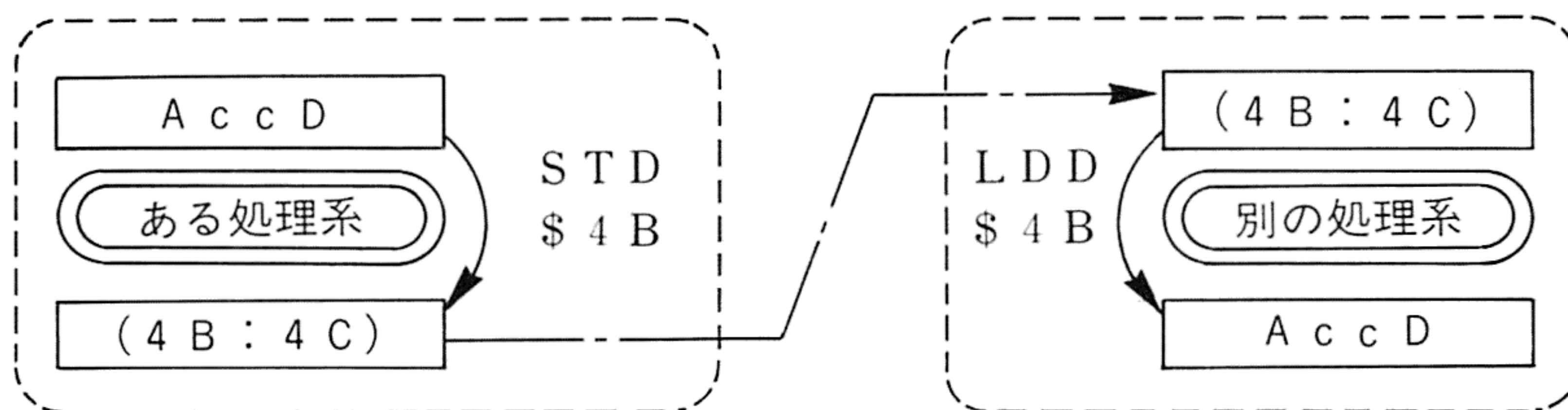



図２・３・２　対サブルーチン情報受渡用レジスタの使用例

2－3－3　メインルーチン・プロンプト表示部

アドレス　＄８Ｅ６３，＄８Ｅ７２（～＄８Ｅ８７）

機　能　「Ready」などのプロンプトを表示して，コマンド待ちの状態であることを明示する．

入力情報　Ｘレジスタに「Break」，「Abort」，エラーメッセージ(４１種)のいずれかの先頭番地（実際には〝先頭番地マイナス１〟が入る；プリントアウトルーチン＄９ＢＤＢを使用するために）を持ってエントリ（ただし，＄８Ｅ６３のみ．＄８Ｅ７２からエントリすると，入力情報は要らず，「Ready」のみ出力される．

復帰情報　復帰せず………（００ＢＦ）をクリアしてテキスト作成部へ進む．

ＷＯＲＫ　（４７：４８）＝直前に実行中だった行番号が格納されている．（ダイレクトモードの場合，＄ＦＦＦＦになる）

（００ＢＦ）　＝ファイル番号

（０５ＡＣ）　＝「プリンタＯＮ」フラグ（第４章参照）

（０５ＢＥ）　＝プリンタ未改行指示フラグ

ＳＵＢ　＄９Ｂ５０──►キャリッジリターンおよびラインフィードを（ＢＦ）のファイルに出力（ここでは画面に出力）

＄９ＢＤＢ──►プリントアウトルーチン；（Ｘ＋１）番地からの文字列をＮＵＬＬ（＄００）または引用符（＄２２）が検出されるまで（ＢＦ）のファイルに連続出力

＄Ｂ６０Ｅ──►▼In␣行番号▼出力ルーチン；出力される行番号は上記のワークレジスタ（４７：４８）内のもの

＄Ｄ６４Ａ──►プリンタにキャリッジリターンおよびラインフィード（改行）を出力するルーチン

解　説　メインルーチン・プロンプト表示部は，大きく分けて，ホットスタートブロック（＄８Ｅ６３～＄８Ｅ７１）とコールドスタートブロック（＄８Ｅ７２～＄８Ｅ８７）の２つに分かれます．

前者は，まず，サブルーチン＄９Ｂ５０によって画面改行を行ったのち，「Break」，「Abort」もしくはエラーメッセージのいずれかを出力しますが，これは中断時の状況によって決まるものであり，エントリの際，Ｘレジスタに情報を蓄えてくるものです．なお，この出力には＄９ＢＤＢのプリントアウトルーチン［ワ

ークレジスタ（ＢＦ）で示されるファイルに出力］を使用します．次に，中断時の実行モードを（４７：４８）によって調べ，（４７＝４８）＝＄ＦＦＦＦである場合，つまり，ダイレクトモードだった場合にはそのままコールドスタートブロックへと進みますが，それ以外の場合には，（４７：４８）に入っている行番号をサブルーチン＄Ｂ６０Ｅで（例えば，「Break In ５１２０」などという具合に）出力してからコールドスタートブロックへと進みます．

後者のコールドスタートブロックは，ＢＡＳＩＣ作動時には必ず一度は通らなければならないブロックであり，簡単な初期設定と〝Ready〞出力を行います．まず，**ファイル番号レジスタ（ＢＦ）**と**プリンタＯＮフラグ（０５ＡＣ）**をクリヤした後，**フラグ（０５ＢＥ）**をテストします．御存知の通り，プリンタというのは，改行が出力された時点で初めて，プリントバッファ中の蓄積された一行をまとめ印字するわけですから，例えば，直前に実行されたＬＰＲＩＮＴ文が〝,〞（コンマ）や〝;〞（セミコロン）で終わっていて改行がなされていない場合，まだ印字されていない文字などがプリントバッファ内に残っていることもあり得ます．そこで，この様に改行がなされていない場合には，未改行を指示するフラグを立てておき，実行終了時（もしくは中断時）にこのフラグをチェックして，未改行であった場合にはプリンタに改行を出力（この動作を行うのがサブルーチン＄Ｄ６４Ａ）し，プリントバッファ中の文字をはき出させてやればよいのです．このフラグが（０５ＢＥ）であり，未改行の際には＄ＦＦとなっています．

続いて，「Ready」プロンプト出力を行います．この動作にも，＄９ＢＤＢが用いられています．この後，＄８Ｅ８８から始まるテキスト作成ルーチンに制御を渡します．

なお，「Break」メッセージ先頭（－１）は＄８Ｄ６０番地［～＄８Ｄ６８］
「Abort」　〃　は＄Ｄ６ＡＣ番地［～＄Ｄ６Ｂ４］
「Ready」　〃　は＄８Ｄ５６番地［～＄８Ｄ６０］
となっています．

また，４１種類のエラーメッセージ群は，＄Ａ４４６～＄Ａ７０Ｃ番地のエリアに格納されています．

２－３－４　メインルーチン・テキスト作成部(テキスト作成ルーチン)

アドレス　＄８Ｅ８８～＄８Ｆ１Ｂ　（ループ形成）

機　　能　コンソールもしくは外部ファイルからプログラムを読み込んで，中間言語のテキストを作成する．また，ダイレクトコマンドが入力された場合には，ダイレクトモードであることを宣言して解読実行部（解読実行ルーチン）へ分岐する．

入力情報　（００ＢＦ）　＝プログラムの読込み先を示すファイル番号［これが＄００であればキーボード入力による入力を，それ以外（＄０１～＄１１）ならば外部ファイルからの読込みを表す］．

ＷＯＲＫ　（１３：１４）＝「作成中の**行の長さ**［バイト数］」プラス５の値が入ります．

（００ＢＦ）　＝ファイル番号レジスタ

（００Ｃ０）　＝入力データ終了フラグ

（３３：３４）＝テキストエリア先頭番地

（３５：３６）＝単純変数エリア　〃　［テキスト終了＋１］

（４７：４８）＝実行中の行番号

（４９：４Ａ）＝ＣＯＮＴ行番号（ここでは一時退避用レジスタとして用いられている）

（４Ｂ：４Ｃ）＝対サブルーチン情報受渡し用レジスタ

（５Ｆ：６０）＝逆向ブロック転送・転送先下限アドレス

（６１：６２）＝　〃　・転送源(もと)下限

（６３：６４）＝　〃　・転送先上限［出力情報］

（６９：６Ａ）＝　〃　・転送源(もと)上限

※１行翻訳ルーチン内では（６９）単独で用いられることもあります．

（Ａ２：Ａ３）＝１行消去・先頭アドレス
（Ａ４：Ａ５）＝１行消去・最終アドレス
（Ａ６：Ａ７）＝ピリオド行番号
（以上３項目：１行消去ルーチン［＄９ＦＤ３］用）

（Ｄ９：ＤＡ）＝汎用読込みポインタ

（０３３８～０４３Ｂ）＝現在編集中の行番号＋テキストバッファ

ＳＵＢ　＄Ｄ９４Ａ　→ＡＵＴＯ編集中の処理ルーチン…（１）で詳述

＄Ｄ７Ｆ７　→１行入力ルーチン（ファイル番号レジスタで示され

るファイルから1行入力し，＄０４３D～＄０５３Cの行入力バッファに格納する）…第4章　参照

＄D０１１　→ＬＯＡＤ等の後処理ルーチン…第4章　参照

＄C２８８ ＄C２８C }→ １行翻訳ルーチン［＄C２８８はテキスト書込用，また，＄C２８Cはダイレクト実行用のエントリ］…（4）

＄C６７A　→解読実行ルーチン・ダイレクト実行用エントリ
…次節　解読実行ルーチン　参照

＄９１６２　→行番号読込みルーチン(文字列⇨行番号定数)…(2)

＄CDC６　→プロテクトチェックルーチン……（3）で詳述
※プロテクト指定については第4章を御参照下さい．

＄８F１C　→行番号サーチルーチン……(5)で詳述

＄９FD３　→１行消去ルーチン（ＤＥＬＥＴＥ処理系サブルーチン；実際には「1行だけ」とは限らない)… **解説** 参照

＄C７３２　→テキストエリアのリンクポインタ修正ルーチン
（部分修正用エントリ）……(6)で詳述

＄８F４B　→ＮＥＷコマンド処理ルーチンの一部で，変数消去や文字配列作業領域初期化を始め，様々な初期化を行う．…３－１　ＮＥＷコマンド処理ルーチン　参照

※(1)～(6)については，この節で順を追って説明していきます．その他については，各部分の説明を御参照下さい．

終了条件　テキスト作成ループ（＄８E８８⇄＄８F１B）からの脱出が行われるのは，次の4つの場合に限ります．

（１）　外部ファイルからの入力終了時⇨＄D０１１に分岐

（２）　ダイレクトコマンド実行を受け入れた場合⇨＄C６７Aに分岐

（３）　入力時・翻訳時のエラー発生⇨エラーコードを持って＄８DD１へ

（４）　ＡＵＴＯ処理がキャンセルされた場合⇨＄８E７２プロンプト表示部へ

復帰情報　各終了条件

（１）では，特に明示的な情報はありません．

（２）では，ポインタ（D9：DA）にテキストバッファ先頭番地

（＄０３３Ｃ）をセットして，解読実行ルーチンへの引き継ぎを行います．

（３）では，エラーコードをＢレジスタ中に格納して，エラー処理ルーチン(＄８ＤＤ１～＄８Ｅ６２)への引き継ぎを行います．

（４）では，現在の編集中行番号（０３３Ａ：０３３Ｂ）が復帰情報に当り，この値はＡＵＴＯ編集処理ルーチンで，ピリオド行番号に代入されます．

解　説　前述した様に，＄８Ｅ８８～＄８Ｆ１Ｂのテキスト作成部は，全てのＢＡＳＩＣプログラム・中間言語テキストを作成するルーチンであり，ＦＭ－７上の他のどの場所でもテキストの作成は行われません．アスキーセーブされたプログラムに対する**ＬＯＡＤ**や**ＭＥＲＧＥ**，あるいは**ＣＨＡＩＮ**や**ＲＵＮ**“ファイルディスクリプタ”などといったコマンドの処理も，**本体は，テキスト作成ルーチンそのもの**であり，各々のエントリ番地付近にある処理系では単に前処理・後処理が行われているに過ぎません．

さて，テキスト作成ルーチンは，１５０バイトにも満たないバイト数のものですが，その中に様々な範疇のサブルーチン，それも癖のあるサブルーチンを含んでおり，とりわけ，1行翻訳ルーチン（＄Ｃ２８８～＄Ｃ５Ｅ１番地）は，８００バイトを優に越える巨大ルーチンであり，把握するには労力を要します．

それでは，**図２・３・３**を御覧下さい．この図は，テキスト作成ルーチンの構成を表すブロック図です．

この中で使用している各サブルーチンについては，順を追って説明していきます．ただ，図中で特に注意を要する箇所がいくつかありますので，これらについて以下に述べます．まず，＄８Ｅ８Ｅでキャリーフラグによる条件分岐を行っていますが，これは1行入力ルーチン（＄Ｄ７Ｆ７）においてキーボード入力中にブレーク・キーかcontrol－Ｃ，control－Ｘの入力があった場合(この時キャリーフラグがセットされる)，その行の入力をキャンセルして，もう一度初めから入力のやり直しを行うためのものです．また，外部ファイルからの１行入力実行中に入力すべきデータが終わってしまった場合には，ワークエリアのフラグ（Ｃ０）を立てて戻ってきますので，このフラグのチェックを行って＄００でなければ，ＬＯＡＤなどのプログラム読込みの後処理ルーチン（＄Ｄ０１１）へ分岐します．

＄８ＥＤＡで，（１３：１４）に入力行の長さ**＋５**の値を格納するとありますが，この「＋５」の意味はおわかりでしょうか．テキストエリアに格納される際，テキストバッファの一行のステートメントに加え先頭にリンクポインタ（２バイ

図2・3・3　テキスト作成ルーチンのブロック図

$8E88

AUTO処理($D94A)
- AUTOフラグをテスト：$00ならそのままリターン
- AUTO行番号発生・画面出力・その他判断など
- 1行入力を行って$8EB5へ…

行入力をキャンセル（Breakキーなど）

$8E8E番地

1行入力ルーチン($D7F7)
- (BF)のファイルから1行入力：行入力バッファに
- Breakキー、control－C·Xが押されていた時は、キャリーフラグをセットして戻る！

キャリークリヤ

キャリーセット

入力データ終了フラグ(C0)テスト

(C0)≠0 → LOAD後処理部($D011)に分岐

＝0

$D2ルーチンにより、バッファから1文字読み込む(数字→キャリーセット)

ダイレクト処理中であることを宣言〔(47：48)に$FFFFを代入する〕

キャリークリヤ〔数字でない〕なら → ファイル番号(BF)チェック → 1行翻訳($C28C) → 解読実行部($C67A)に分岐

Direct In File エラー発生($8DCF)

キャリーセット(数字)なら

$8EAF番地

行番号読込み($9162)

プロテクトチェック($CDC6)

バッファから数字列を読み込み，2バイトの行番号定数に変換して（4B：4C）に格納

行番号が63999を超えたら → Syntax Error‼

$8EB5番地

（4B：4C)の行番号――(33A：33B)に格納

$8EC3番地 → 空白▼␣▼の取扱い

1行翻訳($C288)；結果はテキストバッファ(33C～43B)に格納される

$8EDA番地 → (13：14)に「入力行の長さ＋5」を格納

行番号→(4B：4C)として

行番号サーチ($8F1C)

（4B：4C)に格納された行番号に等しい行番号をテキストから探し当て、そのアドレスをXレジスタおよび(69：6A)に‼ 見つからなければキャリーセット

$8EE2番地 → Xレジスタ〔編集中のアドレス〕を(49：4A)に退避

キャリーセット

編集中のものと同じ行番号を持つ行が無ければ「消去」をスキップ

キャリークリア

Xレジスタの値→(A2：A3)〔消去先頭〕；(X)の値〔リンクの値〕→(A4：A5)〔消去最終〕

1行消去($9FD3)

同時に(A2：A3)→(69：6A)に格納

ブロック転送ソース上限

$8EEF番地 → テキストバッファの1文字目($033C)の読込み

($033C)＝$00

空文（リターンキーのみが押された時）なら、1行消去のみ行う

≠$00

$8E44～8F02

(13：14)で示される値だけ後方のテキストをずらす

同時に、編集中の行番号→ピリオド行番号(A6：A7)

テキスト終了(35：36)→ブロック転送ソース下限(61：62)

(61：62)＋(13：14)→ブロック転送·転送先下限(5F：60)

として、逆向ブロック転送($8D93)

を行う、最後、転送先上限→(63：64)に代入

$8F06～8F0D → テキストバッファの1行を、テキストエリアの、隙間を空けた場所に挿入する

変数消去などの初期化($8F4B番地；NEWルーチンの途中から…)

$8F14番地 → (49：4A)からXレジスタに、退避させておいた「編集中のアドレス」を読み出す

Xレジスタが示す番地以降のテキストエリアのリンクポインタを修正する($C732)

$8F1B番地

(振出に戻る)

ト）と行番号（２バイト）；末尾に行の終りの区切文字ＮＵＬＬ（＄００）の１バイトが付加され，計5バイトだけ本文よりも長くなります．この５バイト分が「＋５」に当るわけです．最後に，＄８ＥＥＣで呼び出している**1行消去ルーチン**（＄９ＦＤ３）；＄８Ｆ００で呼び出している逆向ブロック転送（＄８Ｄ９３）；＄８Ｆ０６～８Ｆ０Ｄのテキスト插入部の３つのブロック転送について，図示しながら関連づけて説明します．これら３つのブロック転送は，組にして使うことを頭に入れて設計されており，その様子は図**２・３・４**に示す通りです．

（１）　ＡＵＴＯ編集中の処理ルーチン

アドレス　＄Ｄ９４Ａ　（＄Ｄ９４４～＄Ｄ９７Ｆ）

機　　能　ＡＵＴＯ編集中フラグ（００９Ｄ）をチェックし，クリア（ＡＵＴＯ編集がかかっていない）の場合はそのままリターン，セットされてる場合は、行信号の自動発生などのＡＵＴＯ編集の１ステップを実行する．

入力情報　（００９Ｄ）　＝ＡＵＴＯ編集実行中フラグ
（９Ｅ：９Ｆ）＝現在編集中のＡＵＴＯ行番号
（Ａ０：Ａ１）＝ＡＵＴＯ行番号の増分（刻み幅）

※これら3つの情報は，ＡＵＴＯコマンド処理ルーチンで設定されます．

復帰情報　（００９Ｄ）　＝次のＡＵＴＯ編集を続けるか否かの情報
（9Ｅ：９Ｆ）＝次のＡＵＴＯ編行番号
（４Ｂ：４Ｃ）＝自動発生させた行番号の値

ＷＯＲＫ　上記の他，ＡＵＴＯ編集がキャンセルされなかった場合には,（Ｄ９：ＤＡ（に行入力バッファ先頭番地「＄０４３Ｄ」をセットしてリターンします．

ＳＵＢ　＄Ｄ９２Ｆ→セットフィールドルーチン［第４章　参照］
＄８Ｆ１Ｃ→行番号サーチ（４Ｂ：４Ｃ）と同一の行番号が見つかった場合にはキャリーフラグをクリア］
＄９ＥＣＣ→（４Ｂ：４Ｃ）に入っている行番号の値を画面に出力する［行番号定数表示ルーチン＄Ｂ６１５を使用］
＄９Ｃ２２──→空白（▾␣▾）を画面に出力［（ＢＦ）ファイルが対象で，ここは画面］
＄Ｄ８０Ａ──→キーボードからの１行入力［Breakキー，contーC　Xでキャリーセット］

図2・3・4　テキスト作成における、3つのブロック転送

①1行消去ルーチン($9FD3)

テキスト
テキスト
(A2：A3)→ DELETE 範囲先頭
消去されるテキスト
(X＋)
(A4：A5)→ DELETE 範囲最終
後方のテキスト
(U＋)
LDA，U＋
STA，X＋
の繰返し

②逆方向ブロック転送($8D93)

転送限上限
(69：6A)・・・・・・→(69：6A)→
この代入は $8EF8で実行
後方のテキスト
(－U)
行の長さ＋5
→(63：64)
新(35：36)
(61：62)→ 転送先下限
(－X)
行の長さ＋5
(61：62)＋(13：14)∴(5F：60)→
転送源下限
この足し算は $8EFCで行われる
LDA，－X
STA，－U
CMPX(69：6A)
BNE ＊－6
というループ

③1行挿入($8F03～8F0D)

テキストバッファ
テキストエリア
$0338→ 04
04
33A→ 行番号
33C→ 1行のステートメント
($0338＋行の長さ＋5) → 00
00
00
(U＋)
逆向ブロック転送におけるXの最終値
(X＋)
←(63：64)
後方のテキスト
LDA，U＋
STA，X＋
CMPX (63：64)
BNE ＊－6
というループ
00
00
00
テキストの終り
←(5F：60)・・・・・・・・・・→ 最終的な(35：36)として代入
〔$8F0E番地で行われる〕

※図中，(0338：0339)に配置されている$0404は，リンクポインタの領域を確保するためのダミーコードです．

解　説　ＡＵＴＯ編集のやり方ですが，ＡＯＴＯコマンド処理系（＄９ＦＥ７～Ａ０１９）では，(９Ｄ)，(９Ｅ：９Ｆ)，(Ａ０：Ａ１）の設定を行ってテキスト作成ルーチンに分岐するだけであり，ＡＵＴＯ編集の本体も，実はテキスト作成ルーチンそのものです．ただ，行番号の自動発生等の付加処理を行うために，このルーチンがあるわけです［この様子をブロック図にしたのが**図２・３・５**です．**図２・３・３**と比較して御覧下さい］．ＡＵＴＯ編集がかかっている場合，このルーチンのリターンアドレスは＄８ＥＢ５となりますが，次のような**キャンセル要因**が成立した場合には，フラグ(９Ｄ) をクリヤしてプロンプト表示部（＄８Ｅ７２）への脱出がなされ，▼Ready▼が表示されます．

（ｉ）編集中の行番号と同一の行番号がテキスト中に既に存在している場合．

(ii）１行入力時にブレークキー，control－Ｃまたはcontrol－Ｘが押された場合．

(iii) 空文（リターンキーのみが押された状態）の入力が行われた場合．

また，次の行番号を計算中に，その値が６３９９９を越えた場合，現在の編集はそのまま続行されますが，フラグ（９Ｄ）をクリアして，**次回のＡＵＴＯ編集**をキャンセルします．

図２・３・５　ＡＵＴＯ編集の様子

$D97B
フラグ(9D)をクリア：AUTO編集キャンセル
$8E72
プロンプト表示部
$D94A
コール(JSR)
$8E88…
フラグ(9D)をテスト
(9D)＝$00ならRTS
AUTO行番号読出し：Xレジスタ←(9E:9F)
AUTO行番号→(4B：4C)に格納
セットフィールド($D92F)
行番号サーチ($8F1C)
1行入力ルーチン($D7F7)
ダイレクトコマンドの選り分け
キャリークリヤ
同一行番号が既にテキストに存在
ダイレクト実行
行番号表示($9ECC→$B615)
空白(▼␣▼)の表示(＄9C22)
1行入力ルーチン(＄8D0A)
行番号読み込み($9162)
プロテクトチェック
キャリーセット
$8EB5…
Break，controlC·Xが押された場合
$D974
入力バッファの1文字目をテスト
(1文字目)＝$00〔NULL〕
(D9：DA)に$43D〔行入力バッファ先頭〕をセット
空文：リターンキーのみが押された
次の行番号計算(9E:9F)←(9E:9F)＋(A0:A1)
行番号が63999を越えた場合
(4B:4C)の行番号→(33A:33B)
(9D)をクリア：次回のAUTO編集キャンセル

(2) 行番号読込みルーチン $9162

アドレス　$9162 ～ $9194

機　能　ポインタ(D9：DA)が指すバッファ位置からの文字列を行番号に変換する［文字列が数字で始まらないときには値として0を返す］.

レジスタ　A, B, X

入力情報　(D9：DA)に, 読み込みの対象となるバッファの位置アドレスを, Aレジスタに, 変換を行おうとする文字列の最初の一文字を, キャリーフラグにはAレジスタの文字（数値の場合）をセットしておく.

※このルーチンは, コール直前に, **$D2ルーチン**もしくは**$D8ルーチン**によって, Aレジスタおよびキャリーフラグを**あらかじめ設定してから**呼出すことを前提としているので注意を要する.

復帰情報　(D9：DA)　=変換した数字列の次の位置
Aレジクタ　=次の文字
キャリーフラグ=Aレジスタの文字が数値ならばセットされる.
（**$D8ルーチン**によって, **空白スキップ**が行われているので要注意）

(4B：4C)　=作成された行番号の値

WORK　上記のワークエリアの他には,
(0011)が作業用レジスタとして用いられる.

SUB　$D8───→汎用読込みルーチン

※このルーチンには明示的なRTS命令はなく, **JMP　$D8**　で呼び出し元にリターンしている.

終了条件　読み込んだ文字が数字でなかった場合(空白▼␣▼の場合も含む), $D8ルーチンを通ってリターンします.
また,(4B：4C)に蓄えられている値が6400(=$1900)以上で, なおかつ数字が続いている場合には,(4B：4C) *10を計算すると, 行番号として許容される値63999を超えてしまいますので, この時点で, Syntax Error ($9089を通り抜けて$92A0番地）に分岐します.

解　説　このルーチンは, 10倍演算のやり方やフラグの立て方などに巧妙なテクニックが用いられており, 参考になる部分も多いかと思いますので, ソースリストを載せておきます. プログラム設計に御役立て下さい.

番地	命令	オペランド	
9162	LDX	#$0000	初期値の代入
	STX	(4B:4C)	
9167	BCS	$916B	――数字が続いていれば，処理続行
	JMP	$D8	――RTSの代わりも兼ねる
916B	SUBA	#$30	読み込んだアスキーコードから 10進数1桁の値を生成する
	STA	$11	
	LDD	**(4B:4C)**	**――格納されている値を読み出す**
	CMPA	#$18	(4B:4C)の値が64000以上であればエラー
	BHI	$9104	
	ASLB		AccD=(4B:4C)*5
	ROLA		
	ASLB		AccD=(4B:4C)*10
	ROLA		
	ADDD	(4B:4C)	
	ASLB		AccD=AccD*2
	ROLA		
	ADDB	$11	アスキーコードから生成した値をAccDに加え合わせる
	ADCA	#$00	
	STD	**(4B:4C)**	**――値の格納**
9183	LDX	(D9:DA)	ポインタ(D9:DA)のインクリメント
	LEAX	$01, X	
	STX	(D9:DA)	
	LDA	, X	フラグの立て方に注目 [数字の時キャリーセット] ※空白スキップ不要のため，$D2ルーチンは使えない
	CMPA	#$3A	
	BCC	$9167	
	SUBA	#$30	
	SUBA	#$D0	
	BRA	$9167	

皆さんの中には，$9183番地以下の処理に，どうして$D2ルーチンを用いないのか不思議に思った方もいらっしゃると思います．この点に関しては，第0章(6)でも述べましたように，$D2ルーチンは空白(▼␣▼)のスキップを行いますから，ここで，もし$D2ルーチンを用いて文字の読込みを行うと，例えば，

「１０␣␣２Ａ…」も行番号「１０２」として扱われてしまい，このようなことを避けるためにも＄９１８３番地以下の似たような処理プログラムを必要とするわけです．

なお，＄９１０４番地には直接エラー発生プログラムは書かれておらず，＄９０８９への分岐命令が書かれています．そしてその＄９０８９にも，＄９２Ａ０への分岐命令が書かれており，結局，＄９２Ａ０でエラーを発生させる［ＬＤＢ ＃＄０２；ＪＭＰ　＄８ＤＤ１（エラー処理ルーチン：Ｂレジスタにエラーコードを入れてジャンプすればよい）の２つの命令で構成される］ことになります．この＄９２Ａ０へのジャンプ命令は「Syntax Error」発生のエントリポイントですので，よく理解しておいて下さい．また，上のような，分岐命令を小刻みに継いでいく方法は，６８系ＣＰＵを使ったコンピュータではよく見受けられます．

（３）　プロテクトチェックルーチン

アドレス　＄ＣＤＣ６　～　＄ＣＤＣＥ　（わずか５バイト）

機　　能　プログラムプロテクトフラグ（００Ｄ１）をチェックして，＄００以外の場合，即ち，プログラムに保護属性が指定してあった場合に，「Protected Program」エラーを発生させる．

レジスタ　エラーが出ない場合は全く使用しない．

入力情報　（００Ｄ１）に，プログラム保護属性の情報

復帰情報　エラー発生の際，Ｂレジスタに「Protected Program」エラーに対するエラーコード６２（＄３Ｅ）が入り，エラー処理ルーチン（＄８ＤＤ１）に分岐

ＷＯＲＫ　（００Ｄ１）＝プログラムプロテクトフラグ

ＳＵＢ　プログラムプロテクトフラグ（Ｄ１）がセットされるのは，インタプリタ内では，ＬＯＡＤ前処理ルーチン（もちろん，ＣＨＡＩＮなどの処理も含まれます）だけに限られます．プログラムの保護属性を指定するのは，ＳＡＶＥコマンドにおいて「Ｐオプション」ですが，これが指定されると，その旨の情報がファイルに書き込まれます．ＬＯＡＤ前処理ルーチンで，ファイルからその旨の情報を拾い上げると，直ちにこの（Ｄ１）フラグがセットされ，以後このプログラムに対する，ＬＩＳＴコマンド・ＥＤＩＴコマンドの使用や，行の削除・修正処理はできなくなります（ただし，ＡＵＴＯコマンドによる付加とＤＥＬＥＴ

Eコマンドによる抹消の２つは行えます．この２つのコマンド処理系ではプロテクトチェックを行っておりません)．このような場合にプロテクトを解除するには，(D１)フラグを直接クリアする以外ありません．

保護属性や「Ｐオプション」についての詳細は，第４章を御参照下さい．

(４)　１行翻訳ルーチン　――中間言語作成工場――

アドレス　＄Ｃ２８８　～　＄Ｃ５Ｅ１

機　　能　行入力バッファに蓄えられているアスキーコードで書かれたプログラムの一行を，中間言語プログラムに翻訳してテキストバッファ（＄０３３Ｃ～＄０４３Ｂ）に格納する．

レジスタ　**すべて使用**する．ただし，ＤＰは＄００を保持，ＳＰ（スタックポインタ）はリターン時に元の値に戻る．

入力情報　＄Ｃ２８８からエントリの場合は，
　Ｘレジスタに，バッファの先頭アドレス－１の値
＄Ｃ２８Ｃからエントリの場合は，
　(D9：DA)に，バッファの先頭アドレスの値
この他，
　(00BF)に，入力バッファに読み込んだプログラムの入力ファイル番号
をキープしておく．これは，プログラムの**地の文**で，**＄80～＄FFの範囲のコード**がきた場合，エラーを出すかどうか判定するために用いられる［外部ファイルからの入力であればそのコードを無視して読込みをそのまま続行するのに対し，キーボード入力であれば「違法な文字コード」として「Syntax Error」を発生させる］．

復帰情報　（Ｄ９：ＤＡ）｝＝テキストバッファの先頭アドレス－１の値
Ｘレジスタ　　　｝　（＝＄０３３Ｂ）
Ｕレジスタ　　＝テキストバッファ上に作成した中間言語プログラムの最終番地＋３
Ｄレジスタ　　＝作成したプログラム１行の長さ＋５
　　　　　　　　（＝Ｕレジスタの値－＄０３３Ａ）

ＷＯＲＫ　（Ｄ９：ＤＡ）＝バッファ読込みポインタ
（００５Ｆ）　＝２バイト中間コード▼ＦＦ××▼指定フラグ

（０ ０ ６ ０）　＝中間コード作成用レジスタ
　　　　　　　　　および，文字列（“定数，’注釈）処理の区切り
（０ ０ ６ １）　＝**変数名綴り内**であることを指示するフラグ
　　　　　　　　　（綴り内――＄ＦＦ；それ以外――＄００）
（０ ０ ６ ２）　＝**DATA文中**であることを指示するフラグ
　　　　　　　　　（ＤＡＴＡ文中――＄０１；外――＄００）
（０ ０ ＢＦ）　＝ファイルバッファ番号（プログラムの入力先を指示；キーボード入力――＄００；外部ファイル――＄１１）　［第４章　参照］
（Ｅ７：Ｅ８）　＝ＬＩＳＴ，ＬＬＩＳＴ行番号処理用**キー・アドレス**格納レジスタ
（０１ＥＦ～０２１６）＝予納語テーブル・ジャンプテーブル索引
（０３３Ｃ～０４３Ｂ）＝テキストバッファ（作成された中間言語プログラムの一行が格納される）
（００７４～００７Ｃ）＝**FAC1**
（０ ０ １ ７）　＝ＦＡＣ１中の**数値の型**（２，３，４，８）
（０ ０ ６ ９）　＝左右カッコのネスティングカウンタ
（４Ｂ：４Ｃ）　＝行番号読込みルーチン（＄９１６２）から結果を引き継ぐレジスタ
（０２７５～０２７７）
（０２９Ｆ～０２Ａ１）　｝＝拡張フック　［ＢＡＳＩＣ拡張用分岐命令を書くためのレジスタ］

ＳＵＢ

＄Ｄ８ルーチン　［数字のとき，**キャリーをセット**して戻る］
＄９４ＦＤ――→アルファベットチェック［ＡccＡの内容判定］
＄９５０Ｆ――→数字チェック　　　　［　〃　］
　※２つとも，条件成立の場合，**キャリーをクリア**して戻る．
＄８５９７――→ブロック転送［ＡccＢのバイト数だけ転送］
＄Ｂ５０Ｂ――→文字列⇨数値・変換ルーチン....第５章　参照
＄９１６２――→行番号読込ルーチン....（２）で既述

終了条件

行入力バッファから読み込んだ文字がＮＵＬＬ（＄００）であった場合，エンディングブロック［解説参照］を通ってリターンします．

解　説

行入力バッファからの**読込みポインタ**としては，ほとんどすべて，**Xレジスタ**が用いられ，（Ｄ９：ＤＡ）を用いた読み込みを行うのは２ヶ所だけです．エントリの違い［＄Ｃ２８８と＄Ｃ２８Ｃの違い］は，このＸレジスタに入

るべき値の与え方の違いであり，＄Ｃ２８８の場合は，Ｘレジスタに読み込み開始番地マイナス１の値を入れてエントリするのに対し，＄Ｃ２８Ｃの場合は，(Ｄ９：ＤＡ）に読み込み開始先頭番地そのものを与えます．このとき与えるアドレスは，何も行入力バッファの先頭番地に限られるわけではなく，未翻訳のプログラムが入っている場所の先頭番地であれば，どこでもよいわけです．ユーザの工夫によって様々な応用がきくルーチンです．

さて，このルーチンのスターティングブロックは＄Ｃ２９３でＣ２９３で，読込みポインタＸの読み出しのあと，拡張フック＄０２７５をコールし，**テキストバッファ先頭番地**＄０３３Ｃを**Ｕレジスタの初期値**として与えます．以上のように，Ｕレジスタはこのルーチンのほぼ全域に渡って**書込みポインタ**として用いられています．

次の＄Ｃ２９４番地から＄Ｃ５Ｅ１までが巨大なループを構成しており（もちろん，間に小ルーチンを含んではいますが)．中間コード・変数綴り・各種定数などを次々に作成しては格納していきます．このループを一度に説明するのは大変ですので，２０のブロックに分割して述べていこうと思います．

〔1〕 初期設定部

アドレス **＄C294 ～ ＄C29B （8バイト）**

機　能 **フラグ（61)(62）および，レジスタ（E7：E8）の4バイトをクリアする．**

解　説 このブロックは，最初と，地の文（即ち，注釈文字列や文字列定数の外部）で▼：▼（コロン）を検出した場合のみ通過します．１つの文が始まる時，上のフラグ・レジスタを初期状態にする働きを持つブロックです．

フラグ（６１）は，現在，読込みポインタＸの読み出している文字列が，**変数名綴り**を構成していることを指示する［このとき＄ＦＦが入る］ものです．また，フラグ（６２）は，現在読み出している文字列が，**ＤＡＴＡ文中**に記述される一連のデータを構成していることを表します．

レジスタ（Ｅ７：Ｅ８）については，[**２０**] で詳述します．

〔2〕 読込み・分析部

| アドレス | $C29C ～ $C2C7 (44バイト)

| 機　　能 | 読込みポインタXの指す位置から1文字をとり出してAレジスタに格納し，その内容や現在のフラグの状態に応じて，どの処理ブロックへ分岐すればよいかを判定する．このとき，ポインタXのインクリメントも同時に行う．

| WORK | (60)＝中間コード作成用レジスタ
(61)＝変数名綴り内であることを表すフラグ
(62)＝DATE文中であることを表すフラグ

| S U B | $C490──→アルファベットテスト・サブルーチン
(アルファベットチェックルーチン**$94FD**をサブルーチンとして使用している)
$950F──→数字チェックルーチン

※これら3つのサブルーチンについては，[14] **アルファベットテスト・サブルーチン**の項でまとめて説明しますので，随時そちらを御参照下さい．

| 解　　説 | まず，「LDA　，X＋」で読み込みを行い，**NULL**（$00：入力行の終りを表す）であれば，[3] のエンティング・ブロックへと分岐します．

NULLでない場合，**フラグ（61）**を見て変数名綴り内であるかどうかを判定し，変数名綴り内であれば，**予約語検索を行わずに**格納のみを行います．ただし，$C490や$950FなどによってAレジスタのコードがアルファベットでも数字でもない場合(変数名として用いることができない文字コードの場合)，フラグ（61）をクリアして変数名綴り外の処理（予約語検索など）を行います．

次に，Aレジスタのコードが**空白▼␣▼**（$20）かどうかを調べ，空白であればやはり格納のみを行います．空白は，変数名綴りや予約語綴りの間に誤って割り込まない限り，いくつあってもかまわないので配置を工夫して下さい．

以上に該当しない場合，Aレジスタのコードを**（60）に格納**しておきます．

四番目に，Aレジスタのコードが引用符▼"▼（$22）かどうかをチェックします．もし，引用符であれば，[6] の中の**$C2FC**にエントリします．

最後に，**DATA文中フラグ（62）**をテストし，DATA文中であれば，やはり格納のみを行います．この際，**コロン▼：▼**であった場合には，格納後，前述の［1］**初期設定部**に戻り，フラグをクリアします．

今まで述べたうちのいずれでもなかった場合，および，変数名綴りフラグが改

めてクリヤされた場合には，［4］**入力チェック部**へ進みます．

なお，中間コード・数値定数以外の文字コードの大部分は，このブロックの「STA　，U+」によって格納されます．

〔3〕 エンディング・ブロック

アドレス ＄C2C8 ～ ＄C2D6 （15バイト）

機　能 書込みポインタUが指している位置から2バイトをクリアした後，Dレジスタに「行の長さプラス5」の値（すなわち「▼Uレジスタの値▼マイナス＄033A」なる値）を計算し，(D9：DA）に「テキストバッファ先頭番地マイナス1」の値を格納して呼び出し元へリターン（RTS）する．

WORK （D9：DA）――テキストバッファ先頭マイナス1（＝＄33B）

※1行翻訳ルーチンの出力情報の一つです．

解　説 まず,「CLR　，U+」を2回続けます．すなわち，行の終りのNULLが分岐に先立って格納されていますのでこれと合わせて，**最後に＄00が3つ連なる**わけですが，これは，ダイレクト実行の際に実行を止めるためのものでテキストエリア内のプログラムであれば，リンクポインタが格納されるべき箇所です．(ここに，＄0000以外のコードが入ると，インタプリタが「まだプログラムが続いている」と解釈して暴走する恐れがあります)．
例えば，MONコマンドでモニタに入る場合，＄033Cからの内容を見てみますと必ず，▼MON▼の中間コード**＄A2**の後に**＄00**が3つ続いているのがわかります．

この後，Uの値をDレジスタに転送し「SUBD　#＄033A」で,「行の長さ＋5」の値を計算し,「＄033B」をXレジスタおよび（D9：DA）に格納して，RTS命令で呼び出し元へリターンします．

〔4〕 入力チェック部

アドレス ＄C2D7 ～ ＄C2E0 （10バイト）

機　能 Aレジスタに入っているコードが，**＄80～＄FF**の範囲にあるときSyntax Xrrorを発生（＄92A0へ分岐）させる．ただし，外部ファイルから

の読み込みプログラムである場合は，エラーは発生せず，Aレジスタの値を無視して[2]へ戻り，次の読み込みを続行する．それ以外は[5]へ進む．

WORK　(00BF)＝ {キーボード入力時$00
外部ファイルからの入力時$00以外の値}

〔5〕 ▼?▼⇒▼PRINT▼中間コード($B9)変換部

アドレス　$C2E1 ～ $C2E9 (9バイト)

機　能　Aレジスタのコードが**クエスチョンマーク**▼?▼($3F)である場合，これを▼PRINT▼の中間コード($B9)に変換し，格納した後[2]へ戻る．それ以外の場合は[6]の**$C2E9番地**へと進む．

〔6〕 注釈文字列・文字列定数処理ブロック

アドレス　$C2E9 ～ $C2FF (23バイト)

機　能　$C2E9から入った場合，Aレジスタのコードが，**シングルクォーテーションマーク**▼'▼($27)であるかどうかを調べ，そうでない場合は次の[7]へと進む．シングルクォーテーションマークだった場合にはまず，コロン($3A)とシングルクォーテーションの中間コード($8D)を格納し，後の文字列を注釈文字列として，入力行の終りまで(つまり，$00が出てくるまで)格納し続ける．——コロンの自動挿入の意味……解読実行部　参照——

一方，$C2FCからエントリするのは，引用符▼"▼でくくられた**文字列定数**が続く場合で，もう一度，引用符▼"▼が出てくるか，行の終りまで一連の文字列定数として格納し続ける．この際，終了判定補助レジスタとして(60)が用いられる．

WORK　(60)＝注釈文字列処理$00；文字列定数処理時$22

〔7〕 &定数処理ブロック

アドレス　$C300 ～ $C314 (21バイト)

機　能　Aレジスタの内容が▼&▼($26)であった場合，&定数として，

以下数字が続く限りそのまま格納していく（この際，間に空白▼␣▼が入ってもそのまま格納して処理を続ける）．そうでない場合は，［８］へ進む．

SUB　＄９５０６──→数字チェック（Ａレジスタ中の文字コードが数字の場合はキャリーフラグをクリア，それ以外の場合はキャリーフラグをセットして戻る）ルーチン

〔８〕　数値定数の処理ブロック

アドレス　＄Ｃ３１５　～　＄Ｃ３５９　（６８バイト）

機　能　Ａレジスタの文字コードが数字（＄３０～＄３９）である場合に，以後に続く**数字列**（数字［および１つまでの小数点，▼Ｄ▼，▼Ｅ▼］のならび）を**数値定数**として，定数表現の中間言語に変換して格納する．数字でない場合は，次の［９］へ進む．

WORK　（Ｄ９：ＤＡ）＝変換しようとする数字列の先頭番地－１
（＄Ｂ５０Ｂルーチンでポインタとして使用）
（７４～７Ｃ）＝ＦＡＣ１（＄Ｂ５０Ｂルーチンの変換結果の数値が格納される）
（００１７）　＝ＦＡＣ１の数値の型が入る
（整数型２；単精度型４；倍精度型８となる）

SUB　＄９５０６──→数字チェックルーチン……［１４］　参照
＄Ｂ５０Ｂ──→文字列⇨数値変換ルーチン
＄８５９７──→ブロック転送（Ｂレジスタで示される回数だけ，（Ｘ＋）──→（Ｕ＋）の転送を繰返すルーチン）

解　説　**文字列⇨数値変換ルーチン**＄Ｂ５０Ｂを用いて，数値定数の文字列を中間言語に変換するブロックです．文字列⇨数値変換ルーチン＄Ｂ５０Ｂは非常に複雑ですが，使い方は簡単ですので，変換したいと数字列が入っている箇所の直前の番地をポインタ（Ｄ９：ＤＡ）にセットして呼び出すだけでよく，変換結果がＦＡＣ１に，また，その数値型が（１７）に格納されて戻ってきます．

さて，＄Ｂ５０Ｂで変換された数値は，頭に「＄ＦＥ××」をつけて格納されます．「××」には数値型を表す識別子（整数型の場合＄０２，単精度型の場合＄０４，倍精度型の場合＄０８）が入ります．ここで注意しなければならないのは，上位バイトが＄００となるような整定数，すなわち，０～２５５の定数について

は，特に「！」や「＃」の指定がない限り，１バイト節約して格納されます．例えば，定数「１２６」は「ＦＥ０１７Ｅ」，定数「３６」は「ＦＥ０１２４」となります．なお，テキスト上には負定数は存在せず，「－７」などは正定数「７」に単項演算子「－」が付くもの，として評価されます．

〔9〕 コロン・セミコロン検出部

アドレス ＄Ｃ３５Ａ ～ ＄Ｃ３６５ （１２バイト）
機　能 Ａレジスタのコードがコロン▼：▼である場合は，格納した後，［１］に戻り，（６１），（６２），（Ｅ７：Ｅ８）をクリアして次の文字へと進む．また，セミコロン▼；▼である場合には格納して次の文字へ進む．

解　説 コロンは，１つのステートメントが終わることを示す区切り文字ですので，入力バッファから取り出した文字がコロンである場合には，テキストバッファに格納した後，変数名綴り内フラグ・ＤＡＴＡ文中フラグ・ＬＩＳＴ用キーアドレスレジスタの３つをすべてクリアした後，次の文字を続み出します．

〔10〕 省略形キーワードサーチ部

アドレス ＄Ｃ３６６ ～ ＄Ｃ３Ａ３ （６２バイト）
機　能 読込みポインタＸで示される位置からの文字列と，＄Ｃ６ＥＤ～＄Ｃ７２Ｆ（省略形キーワードテーブル）に収められている省略形キーワード群とを，ピリオドを区切りとして順次比較していき，一致した場合はその中間コードを読み取ってテキストバッファの次の位置（Ｕレジスタでポイントされる）へ格納する．一致する文字列が無い場合は，ポインタＸを元の値に戻して，［１１］へ進む．

レジスタ
- Ａ＝インプットバッファから読み込まれる文字
- Ｂ＝各キーワードの中間コード
- Ｘ＝読込みポインタ（インプットバッファ上の読出し位置を示す）
- Ｙ＝省略形キーワードテーブル上の位置を表すポインタ
- Ｕ＝書込みポインタ（テキストバッファ上の格納位置を示す）

S＝**スタック領域**は，読込みポインタXの**退避用**および，文字列の一致判定の際の**作業用レジスタ**として使用される．

WORK　なし

解　　説　まず入口で，ポインタXを既に読み込んでしまった1文字分だけ戻した後，Yに$C6EDを入れます．**$C6ED〜$C72F**のキーワードテーブルには，次のように省略形キーワードとそのコードが入っています．

番地	内容	意味
$C6ED	47，4F，2E，**CD**	▼GO.▼および「TO」の中間コード
C6ED	47，4F，53，2E，**CE**	▼GOS.▼および「SUB」の中間コード
C6F6	52，2E，**88**	▼R .▼および「RUN」の中間コード
⋮	⋮	⋮
C72D	57，2E，**A0**	▼W.▼および「WIDTH」の中間コード

このテーブル上の文字列群と，ポインタX上の文字列とを一文字ずつ比較していくわけですが，文字列同士の一致判定は次の図2・3・6の手順で行います．この際，ポイントとなる命令は,「**CLRA；PSHS　X，A**」と「**ORA,S；STA　，S**」および「**LDA　，S+**」の，5つのスタック操作命令です．6809はZ80などと異なり，スタッフ領域をアクセスするのが非常に容易であるため，このようにスタッフ上にアキュムレータの代わりとなるような領域を確保し演算操作を行う，といった手法が非常に効果的となります．
スタック領域を使用しているため，このブロックでは他に特別なワークエリアを必要としません．

図2・3・6　省略形キーワードの一致判定の様子

```
              →ポインタをインクリメントしながら→
LDA, X+        47  4F  2E(GO.)         47  45  54 (GET)
SUBA, Y   −)  47  4F  2E(GO.)     −)  47  4F  2E(GO.)
              ----------------        ----------------
               00  00  00              00  F6  26
                                                    ピリオドが出た所で
                  すべてのORをとる                   一旦打ち切る
                 (1ビットでも0でなけ
                  ればゼロフラグクリア)
                 [00]                    [F6]
       ゼロフラグが立っていれば      ゼロフラグが立っていなければ
                一致                          不一致
                 ↓                             ↓
ポインタYの番地から中間コードを      最終アドレス（C730)でなけれ
Bレジスタに読み込む                  ば次の文字列比較へ
```

ここで，注意しておきたいのは，「GO.」または「GOS.」が発見された場合で，このときは「87，CD（▼GO▼と▼TO▼中間コード)」，もしくは「87，CE（▼GO▼と▼SUB▼)」を2バイト連続格納した後，[**17**] の**行番号処理ブロック**へ分岐します．それ以外のコマンドのキーワードである場合は，中間コードをAレジスタに蓄えて，[**13**] 内のコマンド後処理部（$C457）へと進みます．

なお，図には示してありませんが，インプットバッファから読み込んだ直後，[**14**]内の**小文字⇨大文字変換ルーチン**（$C495）によって，英小文字は大文字に変換してから比較を行います．

省略形キーワードが見つからなかった場合は，ポインタXの値を元に戻してから，次の［**11**］へ進みます．

〔11〕 記号予約語〔＋，－，＊など〕サーチ処理部

アドレス $C3A4～$C3D0 （45バイト）

機　能 ポインタXからAレジスタに読み込み，英字であれば次の［12］のキーワードサーチへと進む一方，それ以外（英記号）のコードの場合，記号予約語［＋，－，＊，／，∧，¥，>，＝，<の9種類］であるかどうかを調べ，記号予約語であれば，それぞれの中間コードに変換して格納する．予約語でない記号の場合にはそのまま格納するが，コンマ▼，▼であった場合，LIST処理用レジスタ（E7：E8）を調べ，**読込みポインタXと一致すれば，**LISTコンドに付加される行番号オペランドを処理するために［17］の行番号処理ブロマックへと分岐する．それ以外は，次の読み込みを続けるために［2］($C29C）へ戻る．なお，処理中は続込みポインタXと書込みポインタUは退避させておく．

WORK (E7：E8)＝LIST・LLIST行番号処理用キー・アドレス格納レジスタ....［20］　参照

SUB $C490──→アルファベットテストルーチン（同時に，**英小文字⇨大文字変換**も行う）

解　説 ポインタXで示される箇所に入っているコードは数字ではないので(数字は［8］の段階で選り分けられてしまう)，英文字か記号のどちらかであるということになります．ここで，英文字の場合は〔12〕以下の予約語サーチへ

回しますが，記号の場合は演算子かどうかを見究め，演算子記号（これも予約語ですから要注意）であれば中間コードに変換します．この際に参照されるのが，＄Ｃ４０７～＄Ｃ４１８番地に置かれる，下図のような対応テーブルです．

番地	+0	+1	番地	+0	+1
$C407	2B(▼+▼)	−D9(▼+▼の中間コード)			
C409	2D(▼−▼)	−DA(▼−▼　〃　)	C411	5C(▼¥▼)	−E4(▼¥▼の中間コード)
C40B	2A(▼*▼)	−DB(▼*▼　〃　)	C413	3E(▼>▼)	−E5(▼>▼　〃　)
C40D	2F(▼/▼)	−DC(▼/▼　〃　)	C415	3D(▼=▼)	−E6(▼=▼　〃　)
C40F	5E(▼^▼)	−DD(▼^▼　〃　)	C417	3C(▼<▼)	−E7(▼<▼　〃　)

さて，記号（もしくはその中間コード）を格納した後は，［**2**］へ戻って次の文字の処理を繰り返しますが，格納した文字が**コンマ**▼，▼である場合には，特別な処理を加える必要があります．それは，「ＬＩＳＴ（もしくは，ＬＬＩＳＴ）"ファイルディスクリプタ"，行番号１［｛－，｝行番号２］」といった形式の場合，，のあとの数字列を行番号定数として扱うために必要となる処理なのですが，詳しくは［**２０**］の方で説明します．

〔１２〕 基本予約語サーチブロック　――予約語検索・第１弾――

アドレス　＄Ｃ３Ｄ１　～　＄Ｃ４０６　（５４バイト）

機　能　読込みポインタＸが指し示す位置から始まる文字列を，**基本予約語**（ＥＮＤ，ＦＯＲなど８９種のコマンド名；ＡＮＤ，ＯＲなど６種の演算子名；ＳＧＮ，ＩＮＴなど４３種の関数名がこれに当たる）の**キーワード綴り**と比較して，一致するものについては対応する中間コードに変換してAccAに入れ［１３］内の後処理部（＄Ｃ４３３もしくは＄Ｃ４５０）へ進む．基本予約語と一致しない場合には，拡張予約語サーチブロック［１３］（＄Ｃ４１９）へ進む．

ＳＵＢ　＄Ｃ４９５――→**英小文字⇨大文字変換**ルーチン（［１４］内）

解　説　＄０１ＥＦ～＄０２１６のワークエリアは，**表２・３・１**のようなテーブルになっており，現在述べている１行翻訳ルーチンはもちろんのこと，**解読実行ルーチン**や**１行逆翻訳ルーチン**（＄Ｃ１６９～＄Ｃ２８７；ＬＩＳＴ処理系サブルーチン）からも参照される，きわめて重要なテーブルです．
このブロック［**１２**］では直接には関係ありませんが，ＢＡＳＩＣインタプリタ上での予約語の体系を理解するために必要不可欠なテーブルですので，ここに掲載しておきます．

表2・3・1　ワークエリア上のテーブル$01EF～$0216の様子

①DISKモードの時

	番地	+0	+1	+3	対応するキーワードとその中間コード
基本予約語	01EF コマンド演算子名	$68 (104種)	$8890 綴りテーブル	$8ADA ジャンプテーブル	END, FOR, …MOD, ￥, >, =, < $80, 81, …, E3, E4, E5, E6, E7
	01F4 関数名	$2B (43種)	$8A2C 同上	$8816 同上	SGN, INT, ABS, FRE, …, TIME, DATE FF80, FF81, FF82, FF83, …, FFA9, FFAA
DISK用拡張キーワード	01F9 DISKコマンド名	$06 (6種)	$7314 同上	$736F 分岐用処理ルーチン	DSKINI, DSKO$, NAME, FIELD、LSET, RSET $E9, E8, EA, EB, EC, ED
	01FE DISK関数名	$09 (9種)	$733C 同上	$7386 同上	DSKF, CVI, CVS, …, LOC, DSKI$ FFAB, —AC, —AD, …, —B2, —B3
拡張キーワード	0203 拡張コマンド名	$06 (6種)	$8030 同上	$805A 同上	CHAIN, ERASE, LLIST, LPRINT, SOUND, PLAY EE, EF, F0, F1, F2, F3
	0208 拡張関数名	$01 (1種)	$8071 同上	$8077 同上	LPOSのみ $FFB4
ユーザー再拡張用キーワード	020D	$00	$0000	$92A0 (エラー)	・$92A0はSyntax Errorのエントリ番地
	0212	$00	$0000	$92A0 (エラー)	

②ROMモードの時

	番地	+0	+1	+3	対応するキーワードとその中間コード
	01EF	$68 (104)	$8890	$8ADA ジャンプテーブル	END, FOR, ……………, >, =, < (104種類)
	01F4	$2B (43)	$8A2C	$8816 同上	SGN, INT, …………… ,TIME, DATE (43種類)
ダミーコードおよびCHAIN以下の分岐用処理ルーチン先頭番地	01F9	$06 (6)	$801D (ダミー)	$805A 分岐用処理ルーチン	・$801Dは, DISK関係のキーワードがROM時のテキストに用いられている場合にLISTをとった際, 空白▼␣▼を出力するためのダミーテーブルの先頭番地 ・$805A, $8077は下段のコマンドなどの処理ルーチン
	01FE	$09 (9)	$801D (ダミー)	$8077	
	0203	$06 (6)	$8030	$805A (ダミー)	CHAIN, ERASE, …………… ,SOUND, PLAY (6種類)
	0208	$01 (1)	$8071	$8077 (ダミー)	LPOS (1種類)
	020D	$00	$0000	$92A0	
	0212	$00	$0000	$92A0	

+0：各クラスのキーワードの個数が入る箇所

+1：各キーワード綴り文字列テーブルの先頭番地が入る箇所

+3：各処理系のエントリ番地のあるジャンプテーブル先頭もしくは，各処理系への分岐を行うための命令が書かれた処理ルーチンの先頭アドレスが入る箇所

基本予約語の数は，先程［１１］で選り分けた演算子記号を除いても１３８種と非常に多く，省略形キーワードのように全ての文字列と比較していると大変な時間を要しますので，まず，頭文字ごとに仕分けしてからサーチします．それは，

「頭文字のアルファベットによる**見出し**」──→「アルファベット別**索引**」──→

──→「キーワード綴り**文字列テーブル**（辞書の**見出し語**に当たる）」

というように3段構えで行われます．我々が**辞書**を引く場合のように，まず，[あ行]とか[か行]あるいは[A][B]といった見出しに従って頁をめくって位置の見当をつけます．次に，五十音順あるいはアルフッベット順に目的の言葉（見出し語）を捜します．そして，見つけた段階でその言葉の意味を拾い上げる，といった手順をとります．ＦＭ－７のＢＡＳＩＣインタプリタも，**図2・3・7**のような**見出し，索引，文字列テーブル**の３ステップを踏みながら，予約語の検索を行います．

それでは，順を追って説明します．まず，

Uレジスタ＝＄８Ｄ２０＋（Aレジスタの値－＄４１）＊２

＄８Ｄ２０	Aレジスタの値	＄４１	＊２
見出しテーブル先頭番地	文字列の頭文字の文字コード	▼A▼の文字コード	増分の計算

によって，見出しを手繰ります．そしてさらに，

Yレジスタ＝（Uレジスタの指し示す番地の内容）２バイト

によって，「索引のアルファベット別先頭番地」を読み込みます．

ここで，Yレジスタ＝＄００００（頭文字がＪ，Ｑ，Ｙ，Ｚ）の場合は基本予約語ではないとして，次の［**１３**］へ進み拡張予約語サーチを行います．そうでない場合は，Yレジスタの指す番地から再びUレジスタに読み込んで，綴り文字列テーブル上の各文字列の先頭番地を読み込みます．読込みポインタX上の文字列と，Uレジスタで示される番地にある綴りとが一致すれば，「ＬＤＡ　＄０２，Ｙ」によって中間コードを読み込む一方，一致しない場合は，Yを**3つ**インクリメントして区切り＄００が出るまで同様のことを繰り返します（区切り＄００の検出は「ＬＤＡ ，Ｙ ；ＢＥＱ　＄Ｃ４１９」で行う）．＄００が出たら，やはり基本予約語ではないとして，［**１３**］へ進みます．

一つ注意しておきたいのは，索引テーブル上にあるコードは完全な中間コードではないことで，＄８０より小さいものについては＄ＦＦ８０を足してやらないと正式なコードとなりません．このように頭に＄ＦＦが付いた2バイトの中間コードは主として関数名のコードですが，ＩＮＰＵＴ(＄ＦＦＡ６)やＴＩＭＥ（＄ＦＦＡ９),ＤＡＴＥ（＄ＦＦＡＡ）のような，コマンド名・関数名をかねた予約語も含まれています．

図2・3・7基本予約語サーチの様子

①見出し（頭文字のアルファベットによって分類するためのアドレステーブル）

番地									
$8D23	8B72,	8B85,	8B92,	8BC0,	8BDC,	8BFB,	8C0B,	8C15,	8C1C
	[A]	[B]	[C]	[D]	[E]	[F]	[G]	[H]	[I]
$8D35	0000	8C32	8C39	8C55,	8C65,	8C6F,	8C7F,	0000,	8C9E
	[J]	[L]	[L]	[M]	[N]	[Q]	[P]	[Q]	[R]
$8D47	8CBD	8CEB	8D04	8D0E,	8D15,	8D1F,	0000,	0000,	
	[S]	[U]	[U]	[V]	[W]	[X]	[Y]	[Z]	

②索引（頭文字のアルファベット毎に分かれて，文字列アドレス（2バイト）＋中間コード（1バイト）という並びが続く。ただし，$80以下のコードは，2バイトコードであり，この値に$FF80を加えてやることによって中間コードを生成する．$00は終りを示す区切り．）

$8B72　8908, 9D ; 8A17, DE ; 8A32, 02 ; …… ; 8AA5, 21 ; 00
[A]　(AUTO)　(AND)　(ABS)　(ANPORT)　(区切)

$8B85　8972, B4 ; 89CD, C7 ; 89D3　; 89D7, C9 ; 00
[B]　(BEEP)　(BUBINI)　(BNBW)　(BUBR)　(区切)
⋮　⋮

$8D1F　8A1C, E0 ; 00
[X]　(XOR)　(区切)

③綴り文字列テーブル（中間コードの若い順に，予約語綴りが並んでいる．また，各々の綴りの区切を明確にするために，各綴りの最後の文字コードには$80が加えてある．）

番地	データ	内容	分類
$8890	45, 4E, C4	〔END(中間コード$80)〕	基本コマンド名および演算子名
$8893	46, 4F, D2	〔FOR (　//　$81)〕	
$8896	4E, 45, 58, D4	〔NEXT(　//　$82)〕	
$ ⋮	⋮	⋮	
$8908	41, 55, 54, CF	〔AUTO(　//　$9D)〕	
$ ⋮	⋮	⋮	
$8972	42, 45, 45, D0	〔BEEP(　//　$B4)〕	
$ ⋮	⋮	⋮	
$8A2A	BD(3D+80)	〔▼=▼(　//　$E6)〕	
$8A2B	BC(3C+80)	〔▼<▼(　//　$E7)〕	
$8A2C	53, 47, CE	〔SNG (　//　$FF80)〕	基本関数名
$8A2F	49, 4E, D4	〔INT (　//　$FF81)〕	
$ ⋮	⋮	⋮	
$8AD5	44, 41, 54, C5	〔DATE(　//　$FFAA)〕	

さて，読み込んだ中間コードが１バイトコード（＄８０～Ｅ７）であった場合には，［１３］内のコマンド後処理部（＄Ｃ４４３）へ分岐（中間コードはＡレジスタに持ったままで）しますが，２バイトコード（＄ＦＦ８０～ＦＦＡＡ）の場合には後処理は必要ありませんのでそのまま格納して（ただし，実際に格納が行われるのは［１３］内の＄Ｃ４５０～Ｃ４５６番地においてですが），次の文字の処理をするために［２］へ戻ります．

〔１３〕 拡張予約語サーチブロック ——予約語検索・第２弾——

アドレス ＄Ｃ４１９ ～ ＄Ｃ４８Ｆ （１１９バイト）

機　能 読込みポインタＸの指す位置からの文字列が，**拡張キーワード**（ＤＩＳＫモードの場合はＤＩＳＫ用拡張キーワードも含む）であるかどうかを調べ，キーワードであれば，その中間コードをＡレジスタ内に生成（２バイトコードの場合にはＢレジスタに＄ＦＦが入り，両者を連続格納してしまう）して，各コマンドの後処理を行う．一方，キーワードでなければ，**変数名綴り内フラグ**(６１)をセット(「ＣＯＭ　＄６１」によって反転して＄ＦＦにしている）してから，次の文字の読み込みを行うべく，［２］へ戻る．

ＷＯＲＫ (５Ｆ)＝２バイト中間コード▼ＦＦ××▼指定フラグ
(６０)＝中間コード作成用レジスタ
(６１)＝変数名綴り内フラグ
(６２)＝ＤＡＴＡ文中指示フラグ
＄０１ＥＦ～＄０２１６...表２・３・１　参照

ＳＵＢ ＄Ｃ４９０—→アルファベットテスト・ルーチン
（＄Ｃ４９５—→**小文字⇨大文字変換ルーチンを含む**）

解　説 拡張予約語サーチのアルゴリズムは少々難しくなっていますので，図２・３・８によって大体の流れを握んでいただくことにいたしましょう．図はＤＩＳＫモード時の様子を表していますが，ＲＯＭモード時でも手順は全く変わりません．

まず，(５Ｆ）をクリアして（６０）に＄Ｅ８（ＤＳＫＩＮＩの中間コード）を入れ，Ｕレジスタに＄０１Ｆ９を読み込みます．また，Ｂレジスタは比較すべき綴り文字列の個数を表すカウンタとして用いられ，一回の比較が終わる度にデクリメント［同時に（６０）はインクリメント］されています．サーチは，

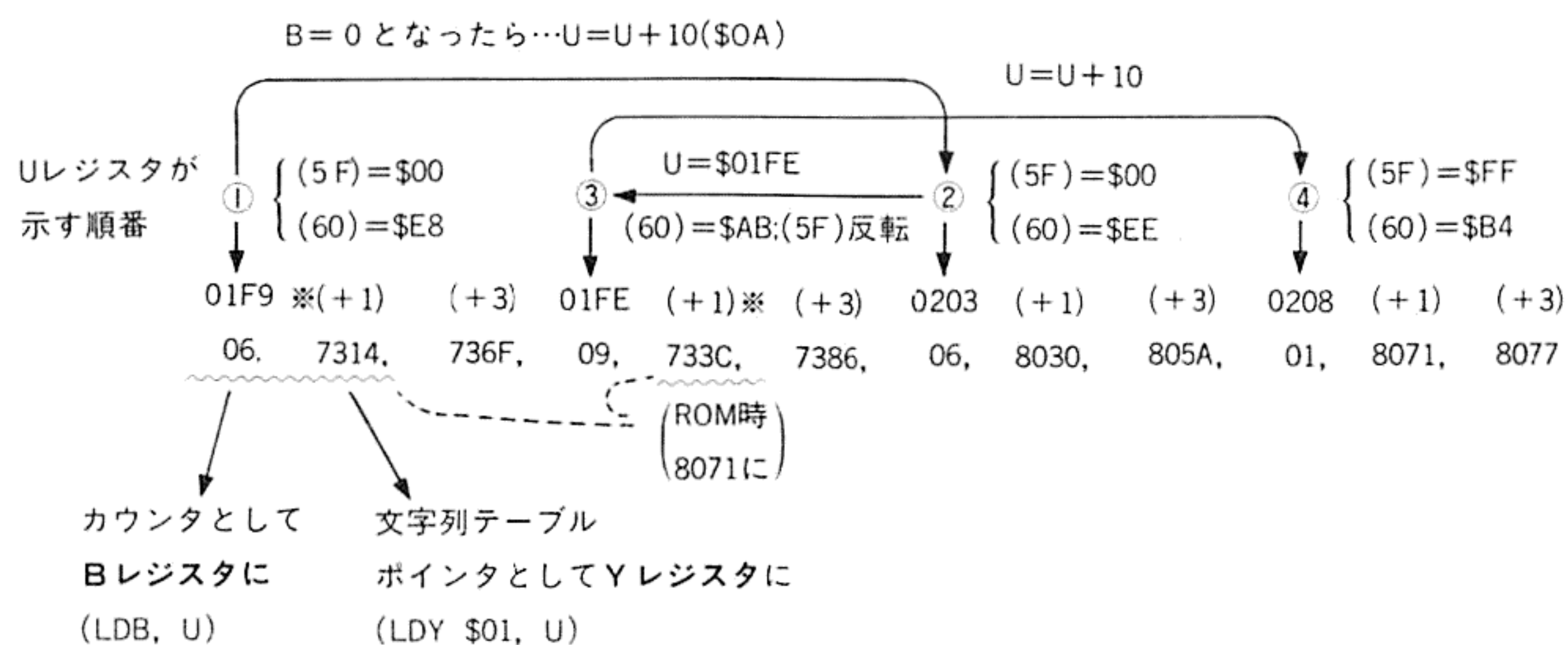
①, ＤＳＫＩＮＩ～ＲＳＥＴ（コード：＄Ｅ８～ＥＤ；Ｕ＝＄０１Ｆ９）
②, ＣＨＡＩＮ～ＰＬＡＹ（コード：＄ＥＥ～Ｆ３；Ｕ＝＄０２０３）
③, ＤＳＫＦ～ＤＳＫＩ＄（コード：＄ＦＦＡＢ～Ｂ３；Ｕ＝＄０１ＦＥ）
④, ＬＰＯＳ（中間コード：＄ＦＦＢ４；Ｕ＝＄０２０８）

の４つのステップに分かれますが，**カウンタＢの値が０になる毎に**次のステップに進むわけです．また，途中で**Ｘ上の文字列**と**Ｙ上**（図参照）**の文字列**が一致した場合，その時点での（５Ｆ）と（６０）が中間コードを与えてくれるのです．

図２・３・８　拡張予約語サーチの様子（ＤＩＳＫモード時）

```
B＝0となったら…U＝U＋10($0A)
U＝U＋10
Uレジスタが示す順番
① (5F)＝$00 (60)＝$E8
③ ← U＝$01FE (60)＝$AB;(5F)反転 ②
② (5F)＝$00 (60)＝$EE
④ (5F)＝$FF (60)＝$B4

01F9 ※(+1)  (+3)  01FE (+1)※ (+3)  0203 (+1)  (+3)  0208 (+1)  (+3)
06.  7314,   736F, 09,  733C,  7386, 06,  8030, 805A, 01,  8071, 8077
(ROM時 8071に)

カウンタとして Bレジスタに (LDB, U)
文字列テーブル ポインタとしてYレジスタに (LDY $01, U)
```

```
テーブルポインタ（Yレジスタ）→→→→……
①7314  44, 53, 4B, 49, 4E, C9, :  44, 53, 4B, 4F, A4 ; … ; 52, 53, 45, D4
(計6種) D   S   K   I   N   I      D   S   K   O   $          R   S   E   T
        《DISK用コマンド名》
        (Y)──→……
②8030  43, 48, 41, 49, CE ; 45, 52, 41, 53, C5 : …… : 50, 4C, 41, D9
(計6種) C   H   A   I   N     E   R   A   S   E          P   L   A   Y
        《拡張コマンド名》
        (Y)──→……
③733C  44, 53, 4B, C6 : 43, 56, C9 : 43, 56, D3 ;…… ; 44, 53, 4B, 49, A4
(計9種) D   S   K   F     C   V   I    C   V   S         D   S   K   I   $
        《DISK関数名》
        (Y)──→……
④805A  4C, 50, 4F, D3
(1種)   ·L  P   O   S
        《拡張関数名》
```

ＲＯＭモードの場合，上図の※を付けた場所には＄８０１Ｄが入っています．そして，＄８０１Ｄ～８０２５は次のような**ダミーコード**テーブルとなっています．

８０１Ｄ　　ＡＯ；ＡＯ；ＡＯ；………；ＡＯ〔＄ＡＯが計９個入っている〕

この＄ＡＯというコードは，空白▼␣▼のコード＄２０に＄８０を足したものであり，読込みポインタＸ上の文字は空白ではあり得ない（空白は［２］の読込み分析部で選り分けられて既にスキップされている）ので，結果的に，**＄Ｅ８～ＥＤ**および**＄ＦＦＡＢ～ＦＦＢ３のＤＩＳＫ関係の中間コード**は生成されないことになります．

なお，このブロックでは，予約語検索の他に各コマンドの後処理を行う場所があります．また，生成した1バイトコードは，＄C457で格納します．

＄C443～C44Fでは，Aレジスタに生成したコードが＄8F(▼ELSE▼の中間コード）であるかどうかを調べ，＄8Fの場合には，コロン▼：▼＄3Aを挿入した後（これは実行時に必要となる区切文字です)，＄8Fをテキストバッファに格納して［17］の行番号処理ブロックへ進みます(▼ELSE 行番号▼の対策)．そうでない場合は，＄C457へ進みます．

＄C450～C456は，2バイトコードが生成されたときに，それを書込みポインタUの指す位置へ格納した後［2］へ戻る，という処理を行う部分です．

＄C457～C460では，生成したコードが＄83(▼DATA▼)であるかどうかを調べ，＄83ならば以下に続く文字列がDATA文のデータであることを宣言（すなわち，データ文中フラグ（62）を立てる）します．

また，＄C461～C467では，生成コードが＄8C(▼REM▼)かどうかをチェックし，そうであれば,［6］の注釈文字列処理ブロックへ分岐します．そうでない場合は,［15］の▼GO▼後処理ブロックへと行きます．

〔14〕 アルフアベットテスト・ルーチン

アドレス　＄C490　～　＄C49F　（16バイト）

機　　能　まず，英小文字⇨大文字変換ルーチン＄C495によってAレジスタ内のコードが，小文字の場合は大文字に換えた後アルファベットチェック・ルーチン＄94FDによって，Aレジスタのコードが英字（＄41～5A）であればキャリーセット，それ以外のコードであればキャリークリアの後戻る．

ＳＵＢ　＄C495(～C49F)──→英小文字⇨大文字変換(内包ルーチン)
※Aレジスタ内のコードが英小文字（＄61～7A）であれば，
大文字(＄41～5A)に変換；このブロックに内包されている．
＄94FD──→アルファベットチェック・ルーチン

これに関連して，アルファベットチェックルーチン（＄94FD～9505）および，数字チェック・ルーチン（＄9506～950E）のソースリストを掲げておきます．

```
$94FD  CMPD  #$41'A'        $9506  CMPA  #$30'0'
       BLO   $9505                 BLO   $950E
       SUBA  #$5B                  SUBA  #$3A
       SUBA  #$A5                  SUBA  #$C6
$9505  RTS                  $950E  RTS
```

〔15〕 ▼GO▼（中間コード＄87）の後処理ブロック

| アドレス | ＄C4A0 ～ ＄C4DA （59バイト）

| 機　　能 | Aレジスタに生成されたコードが＄87（▼GO▼）である場合，その後に▼TO▼または▼SUB▼が続いているかどうかを確認してそれぞれの中間コード（＄CDまたは＄CE）を付加する．この際間に入った空白はそのまま格納する．▼GO▼のあとに▼TO▼も▼SUB▼もない場合には「Syntax Error」を発生させる．なお，格納後，行番号処理ブロック［17］へと進む．一方，＄87でない場合には,［16］へ進む．

| S U B | ＄C495――→英小文字⇨大文字変換ルーチン
＄92A0――→Syntax Errorエントリアドレス

〔16〕 行番号参照コマンド選別ブロック

| アドレス | ＄C4E6 ～ ＄C507 （34バイト）

| 機　　能 | Aレジスタ内のコードが**行番号処理**を必要とするコマンドのコードである場合,［17］(あるいは［19］,［20］)へ分岐する．
それ以外のコードの場合は,［2］へ戻って次の文字の処理を行う．

| 解　　説 | まず，Aレジスタ内のコードが，＄BB（LIST),＄F0（LLIST),もしくは，＄D3（ERL）の，特殊な行番号処理を必要とするコマンドの中間コードである場合の選り分けを行って，それぞれの処理系［19］もしくは［20］へ分岐します．上の3つのコードに当たらない場合は，**行番号参照コマンドコードテーブル**＄C4DB～C4E5（下図）の中のコードと比較していくことにより行番号参照コマンドを選別し,［17］へ飛びます．

$C4DB	9D	88	9A	9E	8B	99	9C	8A	A1	D6	00
(番地)	(AUTO)	(RUN)	(EDIT)	(DELETE)	(RETURN)	(RENUM)	(RESUME)	(RESTORE)	(UNLIST)	(THEN)	(終り)

どのコードでもない場合には，拡張フック＄029F［普段は＄39（RTS）が書かれている］をコールしてから,［2］(＄C29C）へ戻ります．

〔17〕 行番号処理ブロック

アドレス　$C508 〜 $C527 (32バイト)

機　能　読込みポインタXの位置から文字列を，サブルーチン［18］によって行番号定数（もしくはピリオド行番号）に変換して格納する．コンマ▼，▼($2C)やマイナス▼ー▼（$2D⇨$DAに変換）が続く場合は，さらに上の処理を繰り返す．後に続く文字が空白でもコンマでもマイナスでもない場合，［2］の途中$C2A0に入って次の文字の処理へ移行する．

解　説　RENUMコマンドを実行するとテキスト中の行番号もすべて改められますが，これを行うためには，GOTO，RESTOREなどのように後ろに行番号が続くべきコマンドや行番号が続く可能性のあるコマンドについては，定数を普通の数値定数と区別する必要があります．そこで，［16］によって，行番号定数をオペランドに持つコマンドを選別して，後に続く数字の並び（またはピリオド）を行番号定数として特別扱いし，頭に識別子$FEF2を付して格納するわけです（ピリオドの場合はそのまま格納します）．識別子$FEF2は，各コマンドの構文チェックにも用いられます．

ピリオド行番号は，直前に編集・修正などを行った行番号の代用として重宝なものですが．この行番号の値を記憶しておく箇所は，ワークレジスタ（A6：A7）となっています（なお，エラーの際はエラーを起こした行番号が入ります）．

〔18〕 行番号定数処理サブルーチン

アドレス　$C528 〜 $C56A (67バイト)

機　能　読込みポインタXの位置の文字が数字の場合（$9506で判断），$9162ルーチンによって**数字列⇨行番号変数**を行い，$FEF2を付けて格納して戻る一方，ピリオドの場合にはそのまま格納して戻る［この際，数字列もしくはピリオドの前に入った**空白**はそのまま格納］．どちらでもない場合は，ポインタXを一つ戻してリターンする．

WORK　(D9：DA)=$9162ルーチンに対する入力情報
　　　(4B：4C)=$9162ルーチンの出力情報

SUB　$9506──→数字チェック・ルーチン（キャリーフラグ）
　　　$9162──→行番号読込み作成ルーチン

解　説　＄９１６２を呼ぶ前には，**＄Ｄ８ルーチン**による読み込みを必ず行う必要がありますので，＄Ｃ５４１番地付近で「ＬＥＡＸ　－＄０１，Ｘ；ＳＴＸ　＄Ｄ９；ＪＳＲ　＄Ｄ８」という一連の操作をしています（このような操作は［8］における＄Ｂ５０Ｂ**文字⇨数値変換ルーチン**のコール前と，ここの2ヶ所だけです.

なお，注意しておきたいことは，行番号を表す文字列のあとにピリオドが入っていても小数点とは扱われず，例えば,「ＧＯＴＯ　２０．５」と書き込んだ場合，ＬＩＳＴをとると,「ＧＯＴＯ　２０␣．５」,と必ず空白を挿入して警告を発します（この処理を行うのは，＄Ｃ５５Ａ～＄Ｃ５６１）.

〔19〕 ▼ＥＲＬ▼の後処理ブロック

アドレス　＄Ｃ５６Ｂ　～　＄Ｃ５９Ｂ　（４９バイト）

機　能　**▼ＥＲＬ▼のコードの後に比較演算子（＞，＝，＜）が続いていた場合には（間に空白が入っていても構わない），その先に続く文字列に対してサブルーチン［１８］によって行番号処理を施す．比較演算子が続かない場合には，何もせずに［2］(＄Ｃ２９Ｃ番地）戻る．**

解　説　ＥＲＬ関数の場合,「ＩＦ　ＥＲＬ＝＞１８９０　ＴＨＥＮ……」などといった使い方が多く,「ＥＲＬプラス**比較演算子**」の後に続く定数が**ＲＥＮＵＭ**コマンドによって更新される必要があります．そこで,「ＥＲＬプラス比較演算子」の後の数字列を，行番号定数としてＲＥＮＵＭの対象となるようにするのが，このブロックの役割です．なお，ＲＥＮＵＭの対象としたくない場合は（そういうケースがあるかどうか存じませんが‥‥），左辺と右辺を入れ換えて「ＩＦ　１８９０＞＝ＥＲＬ……」などとすれば，定数１８９０はＲＥＮＵＭによって更新されなくなります.

〔20〕 ▼ＬＩＳＴ▼，▼ＬＬＩＳＴ▼後処理部

アドレス　＄Ｃ５９Ｃ　～　＄Ｃ５Ｅ１　（７０バイト）

機　能　**▼ＬＩＳＴ▼，▼ＬＬＩＳＴ▼後に引用符が続く場合（間に空白が場合（間に空白が入っていてもよい），以後の文字列の先読みを行って，引用符の**

対応や左右カッコのネスティングをチェックしながらコンマの検出をし，地の文のコンマであれば，その位置をさすアドレスを（E７：E８）に格納して［２］（＄C２９C）へ戻る．引用符が続かなければ，［１７］の行番号処理ブロックへ分岐する．

WORK　（６９）＝読込みポインタXの一時退避用レジスタ
　　　（D９：DA）＝読名みポインタXの一時退避用レジスタ

SUB　＄D８——→汎用読込みルーチン（空白スキップ付き）

解　説　ＬＩＳＴコマンドは行番号定数をオペランドとして持ち得ますから，後に行番号が続く場合には当然［１７］の処理をしなければなりません．ところが，「ＬＩＳＴ　“(ファイルディスクリピタ)”，行番号」という様に，間にファイルディスクリプタ（以後，ＦＤと略します）が入るという形式もあって，容易には行きません．そのＦＤも，２つの引用符で囲った単純なものばかりであればよいのですが，クォーテーションで始まってさえいれば，どんな文字列式でも出てくる可能性があります．例えば，

「ＬＩＳＴ“ＣＡＳ０：”＋ＬＥＦＴ＄（Ａ＄＋ＩＮＰＵＴ＄（８），８），
　１０００－２０００　」

ということも有り得るわけです．間に入った文字列式については今までと同様に［２］～［１３］のブロックで中間コードに直してやる必要があるわけですが，そのまま［２］に戻ったのでは行番号処理を忘れてしまう場合があります．そこで，ＦＭ－７では，ＦＤの部分の先読みを行っておいて，行番号処理を行うべき場所に（Ｅ７：Ｅ８）という印をつけておくのです．先読みは，行の終りの印（ＮＵＬＬ＝＄００）が出るか，地の文のコンマを発見するまで行われ，コンマ発見の場合はその時の読込みポインタの値を，前述の印として（Ｅ７：Ｅ８）にしまい込みます．いずれの場合にも，読込みポインタXの値を先読み前の値に戻してから［２］(＄Ｃ２９Ｃ）へ戻ります．この際，文字列定数中のコンマや，カッコ内のコンマを拾わないように，引用符の対応や，左右のカッコによるネスティングの様子をチェックしながら検索を行っており，このブロックの大半は，そのためのプログラムで占められています．従って，「ＬＩＳＴ“ＣＡＳ０：”＋（，１５０－４００」などと行うと，「１５０－４００」は行番号として扱われないことになります（いずれにしてもエラーが出るのは明らかです）．

また，次に続く文字が引用符でない場合には，間にＦＤが入り込むことはありません（文法書３．１．３の［形式２］の項［３－８ページ］を御参照下さい）ので，ポインタXの値を元に戻して，直接，［１７］の行番号処理ブロックへ分岐します．

また一方,（Ｅ７：Ｅ８）に先程の印を入れて,［２］以下の処理を続行した場合, コンマが出てくるたびに，ポインタＸの値と（Ｅ７：Ｅ８）の値とを比較するため，両者が一致すると同時に，行番号処理ブロック［１７］へ向かいます（この判断は［１１］の内部の＄Ｃ３Ｃ４番地で行います）.

以上で，１行翻訳ルーチンの解説を終わりますが，最後にサンプルを載せておきます．ＬＩＮＥＩＮＰＵＴ文で読み込んだ一行を，１行翻訳ルーチンによって中間言語に直し，注釈文として領域を確保しておいた１００００行に格納し，ＧＯＳＵＢ文によって実行するというものです.

サンプル

●マシン語ルーチン

```
PAGE   001 (821201,003524) CMD000

00010                                 NAM    CMD000
00020                                 OPT    MEM,NOGEN
00030  6000                           ORG    $6000
00040  6000 9E   D9           CMD000 LDX    $D9
00050  6002 34   10                   PSHS   X
00060  6004 8E   6100                 LDX    #$6100
00070  6007 E6   80                   LDB    ,X+
00080  6009 AE   84                   LDX    ,X
00090  600B CE   043D                 LDU    #$043D   INPUT-BUFFER-ｾﾝﾄｳ
00100  600E DF   D9                   STU    $D9
00110  6010 BD   8597                 JSR    $8597    BLOCK-TFR
00120  6013 6F   C4                   CLR    ,U       NULL:ｸｷﾞﾘﾓｼﾞ
00130  6015 BD   C28C                 JSR    $C28C    ONE-LINE-TRANSRATION
00140  6018 30   01                   LEAX   $01,X
00150  601A 34   50                   PSHS   U,X
00160  601C CC   2710                 LDD    #$2710
00170  601F BD   8F1E                 JSR    $8F1E    LINE-NUMBER-SEARCH
00180  6022 24   03    6027           BHS    *+5      CMD001
00190  6024 7E   9084                 JMP    $9084    Undefined-Line-Number-errror
00200  6027 35   40           CMD001 PULS   U
00210  6029 30   04                   LEAX   $04,X    TEXT-ﾎﾝﾀｲ-LN.10000
00220  602B A6   C0           LOOP   LDA    ,U+
00230  602D A7   80                   STA    ,X+
00240  602F 11A3 E4                   CMPU   ,S
00250  6032 26   F7    602B           BNE    LOOP
00260  6034 35   40                   PULS   U
00270  6036 86   27                   LDA    #$27
00280  6038 A7   82                   STA    ,-X
00290  603A 86   8D                   LDA    #$8D     'ﾉﾁｭｳｶﾝｹﾞﾝｺﾞ
00300  603C A7   82                   STA    ,-X
00310  603E 86   3A                   LDA    #$3A     :
00320  6040 A7   82                   STA    ,-X
00330  6042 35   10                   PULS   X
00340  6044 9F   D9                   STX    $D9
00350  6046 39                        RTS             +++End of CMD001 & Return to ｶ
00360                                 END                ｲﾄﾞｸ･ｼﾞｯｺｳﾙｰﾁﾝ
```

●BASICプログラム

```
10 A$="":A%=0:AD=&H6100
20 LINEINPUT "ｺﾏﾝﾄﾞ ﾊ ?  ",A$
30 A%=VARPTR(A$)
40 POKE AD,PEEK(A%)
50 POKE AD+1,PEEK(A%+1)
60 POKE AD+2,PEEK(A%+2)
70 EXEC &H6000
80 GOSUB 10000
90 GOTO 10
10000 '.......................................................................
..............................................................................
..............................................................................
...............
10010 '.......................................................................
..............................................................................
..............................................................................
...............
10020 RETURN
```

●実　行　例

```
Break
Ready
RUN
ｺﾏﾝﾄﾞ ﾊ ?  X=3.14159265/4:X=SIN(X):PRINT "X=  ";X
X=   .707107
ｺﾏﾝﾄﾞ ﾊ ?


Break In 20
Ready
```

```
10 A$="":A%=0:AD=&H6100
20 LINEINPUT "ｺﾏﾝﾄﾞ ﾊ ?  ",A$
30 A%=VARPTR(A$)
40 POKE AD,PEEK(A%)
50 POKE AD+1,PEEK(A%+1)
60 POKE AD+2,PEEK(A%+2)
70 EXEC &H6000
80 GOSUB 10000
90 GOTO 10
10000 X=3.14159265#/4:X=SIN(X):PRINT "X=  ";X'................................
..............................................................................
..............................................................................
..................
10010 '.......................................................................
..............................................................................
..............................................................................
...............
10020 RETURN
```

２０行のＬＩＮＥＩＮＰＵＴ文を，オープンしたファイルに対して行うように改造すれば，ファイルにアスキーセーブした，任意のコマンド・ステートメントを実行させせることができるなど，工夫によって様々な応用が利くサンプルです．なおその際，プログラムの実行に先立って，ＣＬＥＡＲコマンドによるマシン語ルーチン格納エリアの確保と，マシン語ルーチン（７１バイト）の読込みがなされていなければなりません．

(5) 行番号サーチルーチン

アドレス　＄８Ｆ１Ｃ　～　＄８Ｆ３１

機　能　希望の行番号を持つ行を，テキスト上でリンクポインタを辿りながら捜しあて，その場所のアドレスを**X**レジスタおよび（６９：６Ａ）に入れて返す．該当する行番号がない場合は，その行番号より大きな行番号が出たところで，その行の位置を持って帰る．入力した行番号以上の行番号がない場合は，**テキスト最終アドレス－１**が返される．

レジスタ　Ｄ，Ｘ，Ｕ，ＣＣ（特に**キャリーフラグ**が問題となる）

入力情報　サーチしたい行番号を（４Ｂ：４Ｃ）〔エントリによってはＤレジスタの場合もある〕に入れてコールする．

復帰情報　Ｘレジスタ，および（６９：６Ａ）に，上で述べたテキストアドレスが入る．また，該当行番号の有無によって**キャリーフラグ**が変わり，なかった場合**キャリーセット**；あった場合**キャリークリア**となる．

解　説　テキスト作成ルーチンを初め，ＧＯＴＯ文，ＧＯＳＵＢ文など行番号の参照を必要とするコマンドのほとんどがこのルーチンを用いて，参照したい行番号を持つテキスト行の位置を探索します．探し物が発見されたかどうかは**キャリーフラグ**によって示されるので，「Undefined Line Number」エラーの発生はキャリーセットを見てなされます．逆に，ＡＵＴＯ処理中などでは，自動発生した行番号が発見された場合，そのまま書き込みを続けることは不適当なため，キャリークリアを見ると直ちにＡＵＴＯ処理を打ち切ります．なお，ある特定の番地からサーチを行いたい場合には，サーチ行番号をＤレジスタ，サーチ開始番地をＸレジスタに入れて，**＄８Ｆ２０**からエントリして下さい．ソースリストを載せますので，リンクポインタの重要性やフラグの立て方を参考にして下さい．

```
$8F1C   LDD    (4B:4C)
        LDX    (33:34)
$8F20   LDU    ,X        リンクポインター=0即ち,
        BEQ    $8F2D     テキスト終了ならリターン
        CMPD   $02,X……比較
        BLS    $8F2F     入力行番号以上
                         ならリターン
        LDX    ,X
        BRA    $8F20

$8F2D   ORCC   #$01
          (キャリーフラグセット)
$8F2F   STX    (69:6A)
        RTS
```

〔行の構造〕

```
+0 +1 +2 +3 +4
[リンク][行番号][本 … 体] 00 [リンク] …
```

(6) リンクポインタ修正ルーチン

アドレス	＄Ｃ７３０　～＄Ｃ７４Ｅ

機　能　ＢＡＳＩＣプログラム・テキストのリンクポインタの正しい値を付けていく．なお，エントリによって修正範囲が異なるので注意する．

＄Ｃ７３０→テキスト全部について修正を行う

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　＄Ｃ７３０エントリ──→Ｘレジスタに修正開始番地を正しくセット．
＄Ｃ７３２　〃　──→なし［強いて言えば，（３３：３４）にテキスト開始番地を入れること］．

復帰情報　Ｘレジスタに，**テキスト最終番地マイナス１**が入ってしまう．

ＷＯＲＫ　（３３：３４）＝テキスト開始番地

解　説　リンクポインタの重要性は，**（５）行番号サーチルーチン**のところで理解していただけたと思います．このように，あるデータの塊の特定部分に，次のデータが置かれる場所を示すポインタが入り，最終的にはこのようなデータが次々に積み重って有機的構造体を成す，といったものを，**リスト構造をもつ**ということもありますが，ＢＡＳＩＣプログラムテキストは，最も原始的なリスト構造体の一つであるといえます．

さて，テキストに削除・挿入・修正などを加えて移動が行われると，リンクポインタ内の値は正しくなくなり，作成したリスト構造がこわれてしまうことになります．そこで，文の切れ目（ＮＵＬＬ＝＄００）を辿っていきながら，リンクポインタを修正する必要がありますが，前述したように，＄００があるからといって文の終りとは限らない──すなわち，数値定数内の＄００が存在し得るのでテキストを読んでいって数値定数をスキップする必要が出てきます．この数値定数は必ず，頭が**＄ＦＥ**で始まり，その次の１バイトで**型**を表します．この**型**の下位４ビットは，そのまま以後の使用バイト数を表していますので，これを利用してスキップを行います．なお，終了判断は，リンクポインタの値が**＄００００**に等しいかどうかで行われます．いくら，リンクポインタの値が狂ってしまうとはいえ，テキストの終了を表す目印（＄００００）だけは有効なのです．……

また，誤ってＮＥＷコマンドによって消去してしまったテキストを復活したい場合には，テキスト先頭のリンクポインタを**＄００００以外の値**を入れた後，このルーチン（＄Ｃ７３０）をコールすればうまくいきます．

2－3－5　メインルーチン・解読実行部（解読実行ルーチン）

アドレス　＄C63D ～ ＄C6EC

※ダイレクト実行の際のエントリ番地は**＄C67A**となる.

機　能　読込みポインタ（D9：DA）で示されるテキスト上の中間コードを解読・吟味した後で，各処理系のエントリ番地を求め，各処理ルーチンを呼出し実行する．テキスト終了（**暗黙END**）の判定も行う．

入力情報　読込みポインタ（D9：DA）が正しく設定されていればよい．
この他に入力情報となるのは，OSにおけるsemaphoreに対応する，待機中のBASIC割込みの個数（059A）のみ．

復帰情報　プログラムテキスト終了（暗黙END）時，**キャリーフラグをクリヤ**して，END・STOPルーチンの途中（＄8FE4）に入る．

WORK
(47：48)＝現在実行中の行番号
(4F：50)＝現在実行中のテキスト番地
(00A9)　＝トレースモード・フラグ（TRON／$\overline{\text{TROFF}}$）
(AF：B0)＝スタックポインタの復帰用の値・格納レジスタ
(00B1)　＝エラーRESUMEフラグ
(D9：DA)＝汎用読込みポインタ(ここではテキスト番地が入る)
(059A)　＝待機中のBASIC割込みの個数 (semaphore)
(01F2：01F3)｝ジャンプテーブル先頭番地など（前節
(01FC：01FD)｝（4）の［12］表2・3・1参照)
(0257～0259)＝PENコマンド用拡張フック
(0263～0265)｝拡張フック・普段は**RTS命令**がある
(0266～0268)｝（JMP命令を書くための3バイト）

SUB
＄9195→LET（代入文）処理ルーチン…　**3－8節** 参照
＄9C27→1文字出力ルーチン（JMP　**＄D08E**…第4章）
＄9DC1→MID＄文処理ルーチン
＄B615→行番号定数出力ルーチン
＄BFDD→INPUT処理ルーチン
＄D3C2→BASIC割込み(COM，KEYなど)制御ルーチン
＄DB59→Break キーチェックルーチン　**2－3－6節** 参照
＄E1E4→TIMEコマンド処理ルーチン
＄E291→DATEコマンド処理ルーチン
＄EB4B→SCREENコマンド処理ルーチン

＄8FE4→ENDルーチンの途中のエントリポイント（3－4節）
＄8DD1→エラー処理ルーチン・エントリ番地

解　説　**解読実行ルーチン**の基本的なアルゴリズムは，**テキスト作成ルーチン**のアルゴリズムに比べるとはるかにシンプルで理解しやすいものとなっていますが，このことは実行時のロスタイムを小さくすることと大きな関係があります．すなわち，骨格となる処理は「**（ジャンプテーブル先頭番地）＋2＊（中間コード）」の計算から各処理系エントリ番地の書かれている場所を割り出し，そこからエントリ番地を取り込んで分岐する**という単純なものとなっているのです．

さて，**解読実行ルーチン**は，処理の効率を上げるために，上述の**エントリ番地を計算して分岐する部分**（＄Ｃ６８０～Ｃ６ＤＡ）をサブルーチンの形として持っており，**本体**（＄Ｃ６３Ｄ～Ｃ６７Ｆ）と大きく２つのブロックに別れております．すなわち，**各コマンド処理系ルーチン**は，解読実行ルーチン本体のサブルーチンの一端となっているわけで，ＲＴＳ命令一つでこの解読実行ルーチンに戻ってくることができるのも，このような構造によるものです（**図2・3・9**）．

図2・3・9　解読実行ルーチンと各処理系との連係の様子

(2)　FOR文，WHILE文等のスタック操作コマンドは，リターンアドレスを放棄してJMPで戻る

解読実行ルーチン
$C63D
・BASIC割込のチェック
・ポインタの更新
・構文チェック
$C67A
BSR $C680
$C67F
BRA $C63D

$C680
（中間コード＊2をテーブル先頭に加えて分岐……）
$C6EC
分岐アドレス計算部
コード
エントリ
JMP
各処理系
JMP $C63D
JMP $8E72
RTS命令

(1)　普通のコマンド（LET文，GOTO文など）はRTS命令で解読実行ルーチンに戻る

テキスト作成ルーチン
$8E72
・「Ready」表示
・1行入力
JMP $C67A
ダイレクト実行

(3)LIST，ENDなど実行後に「コマンド待ち状態」となるコマンドは$8E72へ．
※この他，例外的なものとして
(4)　JMP $8E63で「Break」表示を行うもの(STOP文のみ)がある．

それでは，抜粋したソースリスト（**図2・3・10**）で御覧下さい．

図2・3・10 解読実行ルーチンの概略（2の1）

```
本　体        C63D JSR  $0263   …拡張フック
                   JSR  $DB59   …ブレークキーチェックルーチン
BASIC割込 {        LDA  (059A)  } 待機中のBASIC割り込みがあれば，割込み制
処理      {         ：    ：    } 御ルーチン($D3C2)へ分岐
ポインタ  { C64B STS  (AF:B0)…スタックポインタ復帰用の値を格納
更新      {      LDX  (D9:DA)
          {      STX  (4F:50)…テキスト番地の復帰用の値
シンタクス{        LDX  ,X+     } 文の終り($00または▼:▼)でなければSyntax
チェック  {         ：    ：    } Error;▼:▼なら$C67Aに飛ぶ
          { C65D LDD  ,X++     } リンクポインタ読み込み;テキスト終了(リン
次の行へ  {      BEQ  $C6DB    } ク=0)なら，暗黙END処理部($C6DB)へ
移行      {      STD  (47:48)…行番号の格納
          {      STX  (D9:DA)…読み込みポインタ設定
トレース  {        LDA  (A9)    } トレースモードフラグが立っていたら，
モード処理{         ：    ：    } ▼[▼+(行番号)+▼]▼を画面に表示
          { C67A JSR  $D2      …次の文字の読み込み
解読実行  {      BSR  $C680    …各処理系ルーチンへの分岐
          {      BRA  $C63D
```

```
エントリ番地の取込・分岐サブルーチン

C680 BNE   $C683 } 空文（文字コードが$00または▼:▼)
     RTS         } なら本体ブロックへリターン
      ：     ：
C68F CMPA  #$80  } Aレジスタ内の文字コードが中間コードでなければ
     LBLO  $9195 } LET文(代入文)処理ルーチンへ分岐
      ：     ：  }  ※
     LDX   $01F2 }    (01F2:F3)=$8ADA (ジャンプ テーブル 先頭)
C69C ASLA          …A=A*2(この際，最上位のビットが切捨てられ，
     TFR   A, B     値は$00～FEに)
     ABX           …X=(テーブル先頭)+2*(中間コード)
     LDX   ,X      …エントリ番地(分岐アドレス)の取り込み
     JSR   $D2   …次の文字コードを読み込む
     JMP   ,X    …各コマンド処理ルーチンへ分岐
C6B2 JSR   $D2   …(頭がコード$FFで始まっている時)後続コードの読み
                                                          込み
     CMPA  #$9E  } ▼FF9E▼なら，MID$文処理ルーチンへ
     LBEQ  $9DC1 }
      ：     ：
     CMPA  #A0   } ▼FFA0▼なら，SCREENコマンド処理系へ
     LBEQ  $EB4B }
     JMP   $0266…拡張フック
```

図2・3・10　解読実行ルーチンの概略（2の2）

```
暗黙END  C6DB  LDB    (47:48)  ┐ ダイレクト実行中なら、RESUMEチェック
処理部         INCB            │ を行わずに、$C6E8へ行く
               BEQ    $C6E8    ┘
               LDB    #$13…… Error Without Resumeエラーコード
               TST    (B1)     ┐
               LBNE   $8DD1    ┘
         C6E8  ANDCC  #$FE……キャリークリア($8FE4で必要となる)
               JMP    $8FE4……END・STOP・ブレーク処理ブロック
```

本体ブロックでは，ＢＡＳＩＣ割込み処理→ポインタ更新→シンタクス（構文）チェック→新しい行への移行（テキスト終了テストも含む）→トレースモード処理→解読実行の順に処理が進められます．なお，**構文チェック**は，前のコマンドの処理が終了した際に文が正しく終結しているかどうかをチェックするためのものですが，次のコードが**コロン**（▼：▼＝＄３Ａ）の時は，新しい行への移行処理とトレースモード処理の２つをスキップして，すぐに次のステートメントを実行します．また，トレースモード処理において，▼［▼および▼］▼の表示には**１文字出力ルーチン**（＄９Ｃ２７⇨＄Ｄ０８Ｅ；Ａレジスタの文字コードを出力）を，行番号の表示には**行番号出力ルーチン**（＄Ｂ６１５；Ｄレジスタ内の行番号定数を出力）を，それぞれ使用しています．解読実行ルーチンに入る時には常に(ＢＦ)＝＄００となっていますので，出力対象ファイルはコンソール（画面）になります．

分岐用サブルーチン内に※と記した場所では，**中間コードの分類**を行います．読込みルーチンによってＡレジスタ内に取り込んだコードが，

```
┌ ＄８０～＄ＣＢ──→基本コマンドジャンプテーブル(８ＡＤＡ～８Ｂ７１)
│                    より飛び先を求めて各処理系エントリへ分岐
│           ┌ ＄ＣＣ～＄Ｅ７──→Syntax Error（コマンドではない）
└ ＄ＣＣ以上 ┤ ＄Ｅ８～＄ＦＥ──→ＪＭＰ　［０１ＦＣ］（間接分岐）
            │        ┌ 後続　９Ｅ→ＭＩＤ＄文　　Ａ６→ＴＩＭＥ文
            └ ＄ＦＦ ┤ コー　９Ｂ→ＰＥＮ　　　　ＡＡ→ＤＡＴＥ文
         （Ｃ６Ｂ２）└ ドが　Ａ６→ＩＮＰＵＴ文　Ａ０→ＳＣＲＥＥＮ文
```

と分かれます．（１ＦＣ：１ＦＤ）には，ＲＯＭ時**＄８０５Ａ**；ＤＩＳＫ時**＄７３６Ｆ**が書かれており，拡張コマンド・ＤＩＳＫコマンドの分岐アドレスを計算する**延長ルーチン**へ間接ジャンプすることになります．

＄７３６Ｆ
- ＄Ｅ８～ＥＤ──→**ＤＩＳＫコマンドジャンプテーブル**（７３３０～７３３Ｂ）よりエントリ番地を読んで分岐
- ＄ＥＥ以上──→ＪＭＰ　［０２０６］即ち**＄８０５Ａ**へ

＄８０５Ａ
- ＄Ｆ３以下──→**拡張コマンドジャンプテーブル**（８０４Ｅ～８０５９）よりエントリ番地を取り込んで分岐
- ＄Ｆ４以上──→ＪＭＰ　［０２１０］普通Syntax Errorへ

いずれの場合も，Ｘレジスタに**ジャンプテーブル先頭**（＄７３３０もしくは＄８０４Ｅ）を入れ，Ａレジスタから**＄Ｅ８**もしくは**＄ＥＤ**を引いて，**＄Ｃ６９Ｃ**に入り直します．コマンドの種類を問わず，**＄Ｄ２ルーチン**によって**中間コードに続く１文字をＡレジスタに読み込んでから**，各処理系ルーチンへ分岐していることに注意して下さい．一方，ジャンプテーブルは次のようになっています．

図２・３・11　ジャンプテーブルの様子

$８ＡＤＡ	８ＦＤ０，	Ａ２０３，	Ａ２Ｃ７，	９０Ａ０，	……	ＥＢ９Ａ，	Ｅ３ＢＡ（8B71番地）
	END	FOR	NEXT	DATA		KILL	INTERVAL
		(81)	(82)	(83)		（CA）	(CB)のエントリ
$７３３０	７３Ａ１，	７５６５，	７ＣＥ６，	７Ｄ４Ｅ，		７ＤＡ８，	７ＤＡ７（733B番地）
	DSKINI	DSKO$	NAME	FIELD		LSET	RSET
	（コードE8）	（Ｅ９）	（EA）	（EB）		（EC）	（ED）
$８０５Ａ	８０８Ｅ，	８４４８，	ＣＤ８８，	９Ａ７Ｆ，		ＥＢＢＡ，	ＥＣ７２（8059番地）
	CHAIN	ERASE	LLIST	LPRINT		SOUND	PLAY
	（EE）	（EF）	（Ｆ０）	（Ｆ１）		（Ｆ２）	（Ｆ３）

すべてのコマンドについてこの頁に載せることはできませんので，図中のコマンド以外のエントリについては，６－４節を御参照下さい．

それから，ＰＥＮコマンドは現在のところ使用不可能であり，**ＰＥＮ用拡張フック**（０２５７～０２５９）には「ＪＭＰ　＄ＣＦ０Ａ（Devic Unavailableエラー・エントリ）」が書かれていますが，ここに，ユーザの作った任意のマシン語処理ルーチンへのＪＭＰ命令を書き込むことによって「ユーザ定義コマンド」としての意味を持たせることができます．

最後に，**暗黙ＥＮＤ処理部**について触れておきます．テキスト（もしくはテキストバッファ上）のプログラムを実行中に**リンクポインタ**を読み込んで，その値が＄００００であった場合，プログラムテキストは**ここで終わり**となっていますので，実行を中止する必要があります．この場合には，**ＥＮＤルーチン**の途中へ分岐していきますが，この際，オープン中のファイルのクローズは行わないことに注意して下さい．このため，エラー処理ルーチン内でこのような**暗黙ＥＮＤ**となった場合には，特に警告として「Error Without Resume」エラーで終わるよう

なしくみになっています．また，ENDルーチンは，STOP・ブレーク処理をかねているので，両者の区別をするために内部で**キャリーフラグ**のクリヤ／セット（クリヤがEND／セットがSTOP・ブレーク）を行っていますから，暗黙ENDとして途中から入る際にキャリーをクリヤして入る必要があります．以上の操作をしているのが，＄C6DB～C6ECの暗黙END処理部です．

それでは，「10␣TRON // 20␣TROFF」といった簡単なプログラムの実行例で，解読実行ルーチンの働きを追ってみます．テキスト上には次のような中間言語が書かれています（番地は標準ROMモード時の値です）．

＄078F：00：0796：000A：90：00：079C：0014：91：00：0000

これを，「RUN」によって実行させた場合，次のようになります．

まず，＄078Fの**＄00**を見て，初めの行のリンクを読み，次いで行番号を読み込みます．最初は**トレースモードフラグ**が立っていないので，トレース処理は行わず，次のコード**＄90**を読みます．基本コマンドの一つですから，Xレジスタにはまず（1F2：1F3）の値＝**＄8ADA**が入り，続いて，ASLAでAレジスタの値は＄20となります．これがXに加えられて**＄8AFA**となり，この＄8AFAにはTRONコマンド処理系のエントリ番地＝**＄A01A**が書かれていますから，結局，＄A01A番地に制御が移ることになります．RTS命令でTRON処理から戻った後，再び同様の処理が繰り返されますが，今度は，TRONによってトレースモードフラグが立てられていますから，▼［▼，▼20▼（現在の行番号），▼］▼が表示されます．次いで，中間コード**＄91**が読み込まれ，先程と同様にして**＄A01B**番地（TROFF処理系）へ分岐します．最後に，＄079Cのリンクポインタ＝**＄0000**を見て，**暗黙END**へ飛びます．そして，**ENDルーチン**を通って**＄8E72**番地に入り，「Ready」表示となるわけです．このとき，TROFFによってすでに，トレースモードフラグはクリヤされています．

ところで，TRON，TROFFの処理系がどのようになっているのかは次のリストを御覧下さい．

図2・3・12　TRON，TROFF処理ルーチン

```
      TRON (A01Aから入った場合)    ←→    TROFF (A01Bから入った場合)
$A01A  86  4F  LDA  #$4F          $A01B  4F       CLRA
       97  A9  STA  $A9                  97  A9   STA  $A9
       39      RTS                       39       RTS
```

左右をよく見比べてみると，5バイトのみで，2つのコマンドの処理をしていることがわかります．このように，1番地エントリをずらすことにより，フラグの

セット／クリアの２つの動作をまとめてしまうという方法は，よく用いられる高等テクニックですので十分理解しておいて下さい．ただし，アセンブラ泣かせのテクニックであることもお忘れなく!!

2－3－6　Breakキーチェックルーチン

アドレス　＄ＤＢ５９　～　＄ＤＢ６Ｂ

機　　能　ブレーク・キーの押下をチェックし，押下が検出されれば，音関係の初期化をしてBreakルーチンにジャンプする．

レジスタ　Ｄ，ＣＣ

ＷＯＲＫ　（０３１２：０３１３）＝ブレーク・キー押下フラグ（ＦＩＲＱでセット．）

ＳＵＢ　＄０２９０──→拡張用

＄ＥＣ０５──→音関係の初期化ルーチン

終了条件　（０３１２：０３１３）＝０なら，何もしないでReturnする．

（０３１２：０３１３）≠０なら，＄ＥＣ０５を呼び，（０３１２：０３１３）をクリアして＄８ＦＤ７（Breakルーチン）へ飛ぶ．

2－3－7　音関係の初期化ルーチン

アドレス　＄ＥＣ０５～＄ＥＣ１３

機　　能　ＦＭ－７の持つ音関係の初期化をするもので主な動作は次のものです．

①＄ＥＣ４ＤでＰＬＡＹ文が使用するカセット用バッファ（カセット・ＰＬＡＹ兼用）とＰＳＧの初期化をします．

②＄ＥＣ２７～＄ＥＣ２８にＢＥＥＰ␣ＯＦＦ用のＲＣＢが格納されており，これを使用しＢＩＯＳを呼び出してＢＥＥＰ␣ＯＦＦします．

③ＰＬＡＹ文ではＴＩＭＥＲの割込みを使用してバッファ内に蓄えられたデータをＰＳＧに出力しているので，＄ＥＣ１Ｂを呼び，ＴＩＭＥＲ割込みを禁止することで，ＰＬＡＹを実行させない様にします．

④Ｆフラグと Ｉフラグのマスクをイネーブルにします（ＡＮＤＣＣ　＃＄ＡＦ）．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ，ＣＣ　（ＣＣ以外は全て保存）

ＷＯＲＫ　＄００ＤＥ──→ＢＩＯＳジャンプルーチン

＄ＥＣ１Ｂ──→ＩＲＱ割込み制御ルーチン

＄ＥＣ４Ｄ──→ＰＬＡＹバッファ，ＰＳＧの初期化ルーチン

第３章　システム的コマンドとそのサブルーチン

３－１　ＮＥＷコマンド処理ルーチン

――テキスト消去・変数消去・各種の初期設定を行うルーチン――

アドレス　Ｓ８Ｆ３２，３９，４Ｂ，５１，７０，８２，９１～＄８ＦＢＢ

機　能　次のように７つのブロックに分かれ，各エントリに対応します．

（１）　８Ｆ３２～３８→構文チェック／ファイルクローズ（ＮＥＷコマンド）

（２）　８Ｆ３９～４Ａ→テキスト消去など

（３）　８Ｆ４Ｂ～５０→読込ポインタ（Ｄ９：ＤＡ）の初期設定

（４）　８Ｆ５１～６Ｆ→変数消去など（ＣＬＥＡＲコマンドに相当）

（５）　８Ｆ７０～８１→エラー関係の初期化／乱数初期化

（６）　８Ｆ８２～９０→ＳＰの初期値設定／ＣＯＮＴアドレスのクリア

（７）　８Ｆ９１～ＢＢ→文字列作業領域（ＳＡＣ１～１０）初期化など

入力情報　＄８Ｆ９１（７）からエントリする場合には，**Ｘレジスタにリターンアドレスを格納しておく．**

ＷＯＲＫ　解説参照

ＳＵＢ　＄９２Ａ０→Syntax Error エントリ

＄ＣＥ８１→すべてのファイルをクローズするルーチン

＄８０ＥＢ→ＣＨＡＩＮ後処理ルーチン

＄Ｄ４９８→ＢＡＳＩＣ割込(ＣＯＭ,ＫＥＹなど)初期化ルーチン

＄８ＦＣ９→ＤＡＴＡ・ＲＥＳＴＯＲＥポインタ初期化ルーチン

＄９８５０→文字列ゴミ化ルーチン

※「ゴミ化」については，**２―２節**内の**文字列変数**の説明を参考にして下さい．

解　説　各エントリは，ＢＡＳＩＣインタプリタ内では次のように使い分けられています．

＄８Ｆ３２（１）→ＮＥＷコマンド処理の入口

＄８Ｆ３９（２）→ＢＡＳＩＣ起動時（コールドスタートルーチン）およびＬＯＡＤ実行中に**エラー**が発生したりAbortがかかったりした時にコールされます．

＄８Ｆ４Ｂ（３）→ＲＵＮ ↵ の時の初期設定用や，ＤＥＬＥＴＥ処理・ＬＯＡＤ処理が終わった後に呼出されるエントリです．

＄８Ｆ５１（４）→ＣＬＥＡＲコマンドの最終処理ルーチンとして，また，「ＲＵＮ　行番号」やＲＥＮＵＭの処理前後にコールされる場所．

＄８Ｅ７０（５）→ＣＨＡＩＮ後処理ルーチンからのエントリ．

＄８Ｅ８２（６）→Abortルーチンからのエントリ．

＄８Ｆ９１（７）→エラー処理ルーチンからコールされる場所．

そして，あるエントリから入った場合には，それ以後の処理をすべて行います．従って，(**１**)から入った場合は(**１**)〜(**７**)の処理を全部行うのに対して，例えば，(**４**)から入った場合は(**４**)〜(**７**)の処理だけを行う，という具合になります．それでは，各ブロックごとに見てみます．

(１)　構文チェックは，テキスト上のＮＥＷコマンドの一文が完結していてゼロフラグが立っている（即ち，最後の文字が**ＮＵＬＬ**か**コロン**になっている）かどうかをチェックするもので，完結していなければ「Syntax Error」（＄９２Ａ０）を発生させます．実際のＮＥＷコマンド処理が始まるのは**＄８Ｆ３６**からであり，ユーザがマシン語プログラム上で同等の処理を行いたい場合には，この番地にエントリする必要があります．＄８Ｆ３６〜３８には「ＪＳＲ　＄ＣＥ８１」が書かれており，**すべてのファイルをクローズ**します．

(２)　テキストエリア先頭番地（３３：３４）からの２バイト，即ち，一番最初のリンクポインタに**＄００００**を代入し，次の番地を（３５：３６）に格納します（この操作が**テキスト消去**に相当するわけですから，ＮＥＷコマンドによって誤って消去してしまったテキストは，最初のリンクポインタを修正することによってすぐに復活することができます)．さらに，**トレースモードフラグ**（Ａ９）と**プロテクトフラグ**（Ｄ１）をクリアするとともに，**ＵＮＬＩＳＴ行番号レジスタ**（０１Ｅ７：０１Ｅ８）に**＄ＦＦＦＦ**を代入します．

(３)　このブロックでは，**テキスト先頭**（３３：３４）**マイナス１**の値を**読込ポインタ**（Ｄ９：ＤＡ）の初期値として設定しています．行番号もＦＤも指定されていない時のＲＵＮコマンドは，(**２**)〜(**７**)の処理を終えてから，即，ＲＴＳ命令で**解読実行ルーチン**へ飛びますので，この設定は重要な意味を持ちます．

(４)　まず最初に**ＣＨＡＩＮ後処理ルーチン**（＄８０ＥＢ）を呼出しますが，ＣＨＡＩＮ実行中フラグ（０５ＢＣ）が立っていない限り，拡張フック（０２８１〜０２８３）を通じてそのままリターンしてきます．次に，**ＢＡＳＩＣ割込初期化ルーチン**（＄Ｄ４９８）を呼び，**待機中の割込個数レジスタ**（０５９Ａ）のクリア；**割込テーブル**（０７１Ｅ〜０７７７）の初期化；ＰＦキー割込の禁止；タイマ割込初期化を行います．**変数消去・変数名のＤＥＦ型指定初期化**も，このブロックで行います．前者は，(３Ｂ：３Ｃ)・(３Ｄ：３Ｅ）に(３５：３６)の値を代入することによって変数エリア内に登録されている変数・ＦＮ関数の情報を無

効にしており，後者は，単精度型指定子**＄０４**を**ＤＥＦ型指定用テーブル**（０３１Ｆ～０３３８）に格納することによって「ＤＥＦＳＮＧ　Ａ－Ｚ」に相当する初期化を行っています．なお，文法書３－１４頁には「ＣＬＥＡＲコマンドを実行すると全てのＤＥＦ文定義情報は無効になる」と書かれていますが，**ＤＥＦＵＳＲ**文により定義された情報だけは**有効のまま**ですので注意して下さい．

（５） ここでは，エラーＲＥＳＵＭＥフラグ（Ｂ１）およびＯＮ␣ＥＲＲＯＲ␣ＧＯＴＯ行番号（Ｂ４：Ｂ５）をクリアするとともに，ＲＮＤテーブル（０３１８～０３１Ｂ）に初期値として＄８０４ＦＣ５７２（１０進数で表してみると０．８１１６０６５２６３７４８１６９となる）を与えます．

（６） 非常に重要なブロックで，スタックに積み上げられているデータのうち最上位２バイトの**リターンアドレスをＸレジスタに読んだ後**，**ＳＰ初期値**として**文字列領域上限アドレス**（３Ｆ：４０）**マイナス１**を与え，この値を**スタックポインタ復帰用レジスタ**（ＡＦ：Ｂ０）初期値として代入し，同時に，**文字列領域との境界**をはっきりさせておくための区切**＄００**をこの番地に格納します．つまり，リターンアドレス**１つ**を除いた，すべてのスタック情報を破壊してしまうわけで，ユーザがこの処理系〔**（１）～（６）**〕を使用する場合には相当注意を払わなければなりません．このような事情から，ＢＡＳＩＣサブルーチン〔ＧＯＳＵＢ文の支配下〕内にＮＥＷ・ＣＬＥＡＲを置くのは御法度（ごはっと）です．もちろん，何重にもネストしているマシン語サブルーチンからエントリするなど言語道断です!!　スタック初期化の後，**ＣＯＮＴ用再スタートテキスト番地**（４Ｄ：４Ｅ）に**＄００００**を代入します．この操作によってＣＯＮＴコマンドが禁止されます．（ＣＯＮＴ→「Can't Continue」エラーが発生）．

（７） ここでは，**ＳＡＣ**と結合している有効な文字列が存在する場合には，次々に**ゴミ化**して結合を切り離して無効にしていき，**ＳＡＣポインタ**（１Ｅ：１Ｆ）初期値として**ＳＡＣ０**の番地**＄０５７８**を設定します．さらに，**ＤＩＭ文実行中フラグ**（１Ａ），**ＬＯＡＤ（バイナリ）中フラグ**（０２Ｆ３），**ＶＡＬ関数実行中サイン**（Ｅ９：ＥＡ）をクリアした後，ＲＴＳします．**（７）**から入る場合，**（１）～（６）**のような，スタック破壊から受ける制約はありませんが，**Ｘレジスタにリターンアドレスを格納してから**エントリする必要があることに気をつけて下さい．

3－2　RESTORE処理ルーチン

アドレス	$8FBC, $8FC9 ～ $8FCF
機　　能	**DATA・RESTOREポインタ（53：54）の設定を行う．**
レジスタ	A，B，X，U，CC（**キャリー→行番号の有無を知らせる**）
入力情報	Aレジスタ　＝RESTORE中間コードの次の文字コード ゼロフラグ　＝Areg 内のコードがNULL・コロンの時セット **※以上の情報は，解読実行ルーチンから分岐してくる直前で，$D2ルーチンにより設定されます．今後，これらの情報については自明のこととして，いちいち断りませんが，各コマンド処理ルーチンに入った時には既に次の文字コードが読込まれており，その属性によってゼロフラグやキャリーフラグがセットされているということを常に頭に入れておいて下さい．** （D9：DA）＝現在のテキスト番地
復帰情報	（53：54）＝DATA・RESTOREポインタの更新値
SUB	$9A3A→行番号定数の評価ルーチン…後述 $8F1C→行番号サーチルーチン………2－3－4節(5)
終了条件	$9A3Aで評価された行番号に等しいものがテキスト上に存在しない場合には，「Undefined Line Number」エラーを発生させる．

解　説　RESTOREのコードの後にすぐ**NULL**（$00）または**コロン**（$3A）が続く場合には，**$8FC9**番地に飛んで，DATA・RESTOREポインタ（53：54）に**テキスト先頭マイナス1**の値を代入します（NEWルーチンからコールされているのもこの$8FC9番地です）．それ以外の場合には，**$9A3A**で評価した行番号を持つテキストの位置を捜し出して，そのアドレス**マイナス1**の値を（53：54）に格納します．なお，RESTOREのコードのすぐ後に**$FEF2**（行番号定数の識別子）が続いていない場合には，$9A3Aルーチン内で「Syntax Error」が発生されます．

◀ 行番号定数の評価ルーチン　$9A3A ▶

行番号定数の識別子$FEF2があることを確認した後，以下に続く定数を（4B：4C）およびDレジスタ内に読込むルーチンです．識別子$FEF2が続かない場合には，「Syntax Error」を発生させます．

3—3　RUN・GOTO・GOSUB処理系ブロック

アドレス　$9012, $902D, $9039, $9055 ～ $9070

機　能　$9012——RUNエントリ

$902D——GO（中間コード$87）エントリ

$9039——GOSUB処理系先頭

$9055——GOTO処理系（飛先番地計算サブルーチン）

※この他,「ON（式）GOSUB」のためのエントリ（$902F）がある.

入力情報　Aレジスタ　　　　　＝テキストの次の文字コード

スタック最上位2バイト＝解読実行ルーチンへのリターンアドレス

復帰情報　GOSUB処理系では, 図3・3・1の①のようなネスティング情報をスタック上に積み上げた後,「JMP　$C63D」で解読実行ルーチンへ戻る. GOSUBの他, ネスティング情報を作成するものに, FOR～NEXTループ処理系およびWHILE～WEND処理系があり, 比較のためにそれぞれが作成するネスティング情報を図の②および③に掲載しておく. スタック最上位には, 3つのうちどの情報であるかを表す識別子として, ▼SUB▼, ▼FOR▼, ▼WHILE▼の中間コード（それぞれ, $CE, $81, $BE）が置かれていることに注目!!

図3・3・1　スタック上に作成されるネスティング情報

① GOSUB処理系

SP	内容	オフセット
新SP →	$CE（▼SUB▼）	+0
	$00 / $00（GOSUB文サイン）	+1 ☆
	現在の（▼SUB▼がある）テキスト番地	+3
	現在の（▼SUB▼がある）行番号	+5
旧SP →	リターンアドレス（$C67E）	+7

スタック上に7バイトの領域を確保

☆（+1：+2）のエリアは,「ON␣KEY␣GOSUB」等の割込の時に用いられ, 割込テーブルの先頭番地が格納される.

FOR～NEXT系では,「LEAS␣$02, S」によってリターンアドレスを放棄する. 解読実行ルーチンへは「JMP␣$C63D」で戻る.

② FOR～NEXT処理系

SP	内容	オフセット
新SP →	$81（▼FOR▼）	+0
	制御変数の実効アドレス	+1
	FOR文の終りのテキスト番地の値	+3
	制御変数の型	+5
	増分値（STEP値）（指数部）	+6
	（仮数部・上）	
	（〃・中）	
	（〃・下）	
	（符号）	+10
	終値（TO値）（指数部）	+11
	※符号縮退の変数型で格納	
旧SP →	▼FOR▼が書かれている行の行番号	+15
		+17

③ WHILE～WEND処理系

SP	内容	オフセット
新SP →	$BE（▼WHILE▼）	+0
	対応する▼WEND▼があるテキスト番地	+1
	▼WHILE▼が書かれているテキスト番地	+3
旧SP →	▼WHILE▼が書かれている行番号	+5
		+7

WHILE～WEND系でも,「FOR～NEXT」と同様,「LEAS␣$02, S」によって, リターンアドレスを放棄.「JMP」で解読実行ルーチンへ戻る.

解　説　この処理系は，３つのコマンド処理系が同居しているブロックです．順番に見ていきます．

(1)　RUN……まず，次の文字コードが入っているAレジスタをスタック上に退避させたあと，**音関係の初期化**（＄ＥＣ０５）を行います．Aレジスタを復帰させてその値が**＄２２**つまり**引用符**（▼"▼）である場合には，**プログラムロード処理系**（＄ＣＦ２２）へ分岐します．引用符でない場合には，続いて，**全てのファイルのクローズ**（＄ＣＥ８１）を行います．次の文字コードが**区切文字**（**NULL**または**コロン**），即ち，「ＲＵＮ」もしくは「ＲＵＮ：×××」であれば，変数消去などの初期化ブロック（**＄８Ｆ４Ｂ**～８ＦＢＢ）へＪＭＰで飛び，＄８ＦＢＢのＲＴＳ命令でそのまま解読実行ルーチンに入って，テキストエリアのプログラムを最初から実行します．それ以外の時（「ＲＵＮ（行番号）」）には，＄８Ｆ５１の初期設定をコールした後，飛先番地を計算して，「ＪＭＰ　＄Ｃ６３Ｄ」で解読実行ルーチンへ向かいます（この時，スタックポインタＳＰは初期化されている）．なお，＄８Ｆ４Ｂと＄８Ｆ５１の違いは，読込ポインタ（Ｄ９：ＤＡ）に，**テキストエリア先頭マイナス１**の値をセットするかどうかの違いだけです(詳しくは，**３―１節**を見て下さい)．また，飛先番地の計算には,(４)の**ＧＯＴＯ処理系**がそのまま用いられます．

(2)　GO……次の文字コードが＄ＣＤ（▼ＴＯ▼）なら（４）のＧＯＴＯ処理に，＄ＣＥ（▼ＳＵＢ▼）なら（３）のＧＯＳＵＢ処理系に入ります（どちらでも無い場合には，「Syntax Error」を発生）．この際，＄Ｄ２ルーチンによって，読込ポインタ（Ｄ９：ＤＡ）を一つ進めます．

(3)　GOSUB……Ｂ＝４として**メモリフルテスト**（＄８ＤＡＡ）をコールして，８バイトのスタック領域が確保できることを確かめた後,（４７：４８）および（Ｄ９：ＤＡ）の内容を読込んでプッシュし，**図３・３・１の**①に示したようなネスティング情報をスタック上に作成します（これは，ＲＥＴＵＲＮ文に必要な情報となります）．続いて，（４）を用いて飛先行番号を計算してから，解読実行ルーチンへ戻ります（ＪＭＰ　＄Ｃ６３Ｄ）．

(4)　GOTO……飛先行番号を計算するブロックです（ＧＯＴＯ文の場合には最後のＲＴＳで解読実行ルーチンに入ります）．まず，**行番号定数評価ルーチン**（＄９Ａ３Ａ；ワークレジスタ（４Ｂ：４Ｃ）に行番号を格納）を呼び，ついで，**非実行文のスキップ処理ルーチン**（＄９０ＡＢ）によって現在の行の残りを読み飛ばし，次の行の先頭番地をＸレジスタに入れます．ここで，飛先行番号≦現在の行番号（４７：４８）であればテキスト先頭番地から；飛先行番号＞現在の行番号であれば次の行の先頭（Ｘレジスタ）から，**行番号サーチルーチン**（＄

８Ｆ１Ｃ）によって，飛先行の先頭番地を検索し，１を引いて読込ポインタ（Ｄ９：ＤＡ）に格納してリターンします．このような場合分けを行うのは，少しでも検索に要する時間を短縮するためです．なお，行番号サーチルーチンで，**キャリーが立てられて**帰ってきた場合，即ち，該当行番号がテキスト上に存在しない場合には，「Undefined Line Number」エラーを発生（**＄９０８４**に分岐）させます．

3―4　END・STOP・Break処理系ブロック

アドレス	＄８ＦＤ０，Ｄ７，Ｄ９，Ｅ１，Ｅ４　～　＄９００１
機　能	＄８ＦＤ０――ＥＮＤ文エントリ ＄８ＦＤ７――Breakエントリ（Breakチェックルーチンより） ＄８ＦＤ９――ＳＴＯＰ文エントリ ＄８ＦＥ１――ＧＣＵＲＳＯＲなどからのBreak エントリ ＄８ＦＥ４――暗黙ＥＮＤ（テキスト終了）エントリ
レジスタ	Ａ，Ｘ（Ｂ，ＵはＥＮＤエントリのみ使用） Ｓ――スタック復帰用レジスタ（ＡＦ：Ｂ０）により補正 （直前に実行中だった行に入る前の値が復帰される） キャリーフラグ――ＳＴＯＰ・Break／$\overline{\text{END}}$フラグとして使用 ゼロフラグ――――ＥＮＤ文，ＳＴＯＰ文が完結しているかどうかの判定に用いられる
入力情報	＄８ＦＥ１，Ｅ４からエントリする場合には，**キャリーフラグを立てて入るか否かで**，最終的に，「Break」出力（＄８Ｅ６３）となるか「Ready」出力（＄８Ｅ７２）となるかに分かれる．
復帰情報	「Break」出力の場合，Ｘレジスタに「Break」メッセージの先頭位置（マイナス１）の値＄**8D60**が入る．また，ダイレクト実行でなかった場合には，様々な情報がセーブされる．具体的には ＳＰには，復帰用の値（ＡＦ：Ｂ０）が―＄８ＦＥ４を除く― （４９：４Ａ）には，直前に実行した行番号（４７：４８）が， （４Ｄ：４Ｅ）には，直前のポインタ（Ｄ９：ＤＡ）の値が， それぞれ格納される（これらは，**CONT**文で使用される）．
ＷＯＲＫ	上記の他，（４Ｆ：５０）がポインタ（Ｄ９：ＤＡ）の一時退避用として，また，（０５９８）がキャリーフラグ読込用レジスタとして，それぞれ用いられる．一方，Breakエントリ（＄８ＦＤ７）では，ファイル番号レジスタ（ＢＦ）がクリアされる．
ＳＵＢ	＄ＣＥ８１→すべてのファイルをクローズ（ＥＮＤ文エントリ＄８ＦＤ０のみ；Break・ＳＴＯＰ・暗黙ＥＮＤではクローズが行われないので注意する）．
終了条件	ＥＮＤ文・ＳＴＯＰ文が完結してないと**ＲＴＳ**してエラーとなる．

3—5　CONTコマンド処理系

アドレス	$9002　～　$9011
機　　能	(4D：4E)≠0なら，その値を(D9：DA)に入れ，さらに，CONT用行番号(49：4A)を読込んで，実効行番号レジスタ(47：48)に入れた後，解読実行ルーチンへリターンする． (4D：4E)＝0の時は，「Can't　Continue」エラーを発生させる．
レジスタ	B，X
入力情報	(4D：4E)＝CONT用のテキスト番地 (49：4A)＝CONT用の行番号
復帰情報	(4D：4E)＝0→Bレジスタに「Can't　Continue」エラーコード$11を入れて，エラー処理ルーチン($8DD1)へ分岐． (4D：4E)≠0→(D9：DA)に(4D：4E)の内容 (47：48)に(49：4A)の内容を入れて，解読実行ルーチンへ．
WORK	上記の通り

解　　説　**3—4節**で述べたように，STOP文・Breakなどによってテキストプログラムが止まった場合には，その時のテキスト番地(D9：DA)が(4D：4E)に，行番号(47：48)が(49：4A)にそれぞれセーブされますから，この値を読込むことによって実行を再開させることができます．ただし，テキストを修正したり，CLEARコマンドを用いたりするなど，**NEWルーチン**の(1)～(6)のブロックの処理を行った場合，スタックポインタが初期化されてしまい，実行を再開することは不可能となります．この実行再開不可能のサインとして，NEWルーチン(6)ブロックでは(4D：4E)に**$0000**を代入しているわけです．

3－6　RETURN

アドレス　＄９０７１　～　＄９０Ａ７

機　　能　スタック上のネスティング情報を検索（＄８Ｄ６９）して識別子▼ＳＵＢ▼を持った情報を捜し出し，ＲＥＴＵＲＮ処理を行う．なお，このブロック内には，「Undefined　Line　Number」エラーエントリ（＄９０８４）が存在する．また，＄９０Ａ０～９０Ａ７は，ＤＡＴＡ文・ＲＥＭ文・ＥＬＳＥ文などの処理系をも兼ねているが，これらについては，３－７節で述べる．

レジスタ　Ａ，Ｂ，Ｘ，Ｓ（内部ループ放棄のための更新）

ＷＯＲＫ　（ＡＦ：Ｂ０）＝スタック復帰用レジスタ
（この処理系で更新がなされる）
（Ｄ９：ＤＡ）＝読込ポインタ
（４７：４８）＝実効行番号レジスタ
｝ＧＯＳＵＢ文を実行した時の値に復帰する．

ＳＵＢ　＄８Ｄ６９→スタック上のネスティング情報を検索するルーチン
（識別子＄ＣＥ▼ＳＵＢ▼を持つものを捜し出す）
＄９０Ａ８→非実行文のスキップ処理ルーチン

解　　説　**＄８Ｄ６９**ルーチンにより，図**３・３・１の①**のような形をした，スタック上の情報を捜し出して，ＧＯＳＵＢ実行時の（Ｄ９：ＤＡ）および（４７：４８）を復帰させて，さらに，**＄９０Ａ８**ルーチンを用いて，ＧＯＳＵＢの文が完結するまで（即ち，ＮＵＬＬまたはコロンが出るまで）読み飛ばします．「ＧＯＳＵＢ（行番号）」の後ろの**地の文**には，好きな文字列を書いておいてもよいのはこのためです．**割込処理ルーチン**（「ＯＮ␣ＫＥＹ␣ＧＯＳＵＢ␣文などで定義されたサブルーチン）からのＲＥＴＵＲＮ文の時には，さらに，**割込のＳＴＯＰ状態の解除**を行います．なお，「ＲＥＴＵＲＮ（行番号）」の場合には，（Ｄ９：ＤＡ）および（４７：４８）の復帰は行わずに，＄９０５３に飛んで，この４つのデータを読み飛ばし（「ＬＥＡＳ␣＄０４，Ｓ」），**＄９０５５**以下のＧＯＴＯルーチンを用いて**戻り先行番号**と**テキスト番地**を割り出して（４７：４８）と（Ｄ９：ＤＡ）に格納して解読実行ルーチンに制御を移します．

なお，＄８Ｄ６９で検索する際，内側の▼ＦＯＲ▼および▼ＷＨＩＬＥ▼のループ情報は（それらが存在すれば）全て放棄します．また検索終了時，識別子＄ＣＥ以外のコードが出た場合「Return　Without　Gosub」となります．

3—6—1　スタック上のネスティング情報を検索するルーチン

アドレス	＄８Ｄ６９，＄８Ｄ６Ｂ　～　＄８Ｄ９２
機　能	スタック上のネスティング情報を検索し，目的のものを見つけ出す．
レジスタ	Ａ，Ｂ，Ｘ（Ｘはこのルーチンの入口で「ＬＥＡＸ␣＄０４，Ｓ」によって，スタックを指すポインタとなる）
入力情報	（５Ａ：５Ｂ）＝目的とするＦＯＲ文制御変数の実効アドレス スタック上のネスティング情報群
復帰情報	目的とするＦＯＲ文が見つかった場合には，そのループ情報が積み上げてあるスタック位置がＸレジスタに格納され，ゼロフラグが立つ．目的とするＦＯＲ文が見つからなかった場合には，その時のスタック位置にある文字コードマイナス＄８１の値がＡレジスタに入り，ゼロフラグがクリアされる．
終了条件	識別子＄８１（▼ＦＯＲ▼）を持つもので目的とするものが見つからないうちに，＄８１でも＄ＢＥ（▼ＷＨＩＬＥ▼）でもないコードを頭とするスタック情報が現れると，ゼロフラグをクリアしてＲＴＳ．

解　説　ＦＯＲ～ＮＥＸＴループ用のスタック情報には，ループで使用する制御変数の実効アドレスも積まれますので，（５Ａ：５Ｂ）に与えた**実効アドレス**をキーコードとしてスタック情報を検索すれば，目的とするＦＯＲ文ループ情報を捜し出すことができます．例えば，「ＮＥＸＴ␣Ｉ％」に対応するＦＯＲ文「ＦＯＲ␣Ｉ％＝１……」のスタック情報を，制御変数Ｉ％の実効番地を手がかりに捜し出したりするために，このルーチンが用いられます．制御変数名が明示されていないＮＥＸＴ文の場合は，（５Ａ：５Ｂ）＝＄００００としてからこのルーチンに入ってくるようになっており，この場合には，識別子▼ＦＯＲ▼を持った情報が発見された時点でリターンします．識別子▼ＦＯＲ▼を持っていながらアドレスが一致しないものや識別子▼ＷＨＩＬＥ▼を持つものについては読み飛ばしてさらに外側のループに対するスタック情報を調べていきますが，▼ＦＯＲ▼でも▼ＮＥＸＴ▼でもないコード（例えば，▼ＳＵＢ▼）が出てきた時は，Ａレジスタにそのコードから＄８１を減じた値を入れ，ゼロフラグをクリアしてから，ＲＴＳで呼出し側に戻ります．ＲＥＴＵＲＮ文では（５Ａ）＝＄ＦＦとしてエントリし，内側のループ情報を全て読み飛ばして放棄するわけです．

3－7　DATA・REM・ELSE処理と非実行文のスキップ

アドレス	＄９０Ａ０，９０Ａ３　～　９０Ａ７（ＤＡＴＡ等のルーチン） ＄９０Ａ８，９０ＡＢ　～　９０Ｆ０（非実行文のスキップ処理）
機　　能	＄９０Ａ０→＄９０Ａ８と続くＤＡＴＡ型のスキップでは，ＮＵＬＬまたは地の文のコロンが出るまでテキストを読み飛ばす． ＄９０Ａ３→＄９０ＡＢと続くＲＥＭ・ＥＬＳＥ型処理では，テキスト区切文字としてＮＵＬＬだけを認め，これが出てくるまでテキストのスキップを続ける．同時に，▼ＩＦ▼⇔▼ＥＬＳＥ▼ネスティングのカウントも行う．これは，ＩＦ～ＴＨＥＮ～ＥＬＳＥ処理系（＄９０Ｆ１～９１３３）で利用される．
レジスタ	Ａ，Ｂ，Ｘ
入力情報	エントリ番地が最大の入力情報（ＤＡＴＡ型⇔ＲＥＭ・ＥＬＳＥ型）となっている．ＮＵＬＬは必ず区切文字となる．
復帰情報	非実行文のスキップ処理ルーチンでは，Ｘレジスタに読み飛ばしを終えたテキスト番地（区切文字のある番地）を入れてリターンする． 一方，ＤＡＴＡ・ＲＥＭ・ＥＬＳＥ処理系（～９０Ａ７）では，スキップ処理ルーチン（＄９０Ａ８または＄９０ＡＢ）を呼び出した後，Ｘレジスタに入ったテキスト番地をポインタ（Ｄ９：ＤＡ）に格納してリターン（ＲＴＳ）する．
ＷＯＲＫ	（Ｄ９：ＤＡ）＝読込ポインタ （００１１）　＝区切文字用レジスタ（Ｂの裏レジスタとして使用） （００Ｅ６）　＝文字定数内フラグ （００９４）　＝ＩＦ～ＥＬＳＥネスティングカウンタ

解　　説　非実行文のスキップ処理系には，**数値定数内**の＄００や**文字列定数**内のコロンなどを区切文字として拾わないように，**文字列定数内フラグ**（Ｅ６）を反転するブロックなどの特殊処理部分があります．指定された区切文字（ＮＵＬＬとコロンの２種，またはＮＵＬＬだけ）が現れるまでＸレジスタをポインタとして読み飛ばしを続けますが，途中**地の文**で＄８Ａ（▼ＩＦ▼）を発見の際は，カウンタ（９４）をプラス１します．なお，Ｂレジスタに目的コードを入れて＄９０ＡＥを呼べば，任意のコードを区切文字として読み飛ばしが可能です．

３—８　ＬＥＴ文（代入文）処理系

—— 変数サーチ，式の評価 ——

アドレス　＄９１９５　～　＄９１Ｅ７（＄９１Ｂ３以降は文字列代入）

機　能　代入文の左辺の変数の実効アドレスを変数サーチルーチン（＄９５０Ｆ）によって割出し，式の評価ルーチン（＄９２Ｃ３）によって評価された右辺の式の結果を代入する．

入力情報　読込みポインタ(Ｄ９：ＤＡ)＝次の文字のあるテキスト番地

復帰情報
- 数値変数→ＦＡＣ1に評価された結果を変数型に変換して格納する．
- 文字列変数→評価結果の文字列を表すＳＤを変数本体に格納する．

ＷＯＲＫ
- （５Ａ：５Ｂ）＝変数実効アドレス（本体の位置；代入文用）
- （００１７）　＝サーチされた変数の型／ＦＡＣ１内の数値の型
- （７６：７７）＝ＦＡＣの一部だが，文字列型データの場合には評価結果を表現するＳＤの位置のアドレスが入る．
- （３５：３６）＝単純変数先頭番地（テキスト最終プラス１）
- （３Ｆ：４０）＝文字列スタック領域終了（上限）アドレス
- （６Ｄ：６Ｅ）＝ＳＤの位置アドレス一時退避用
- （７Ｅ～８０）＝一時的ＳＡＣ（＄９７９０ルーチンと関連）

ＳＵＢ
- ＄９５０Ｆ→変数サーチルーチン……**３—８—１**節　参照
- ＄９２９４→任意の文字コードチェックを行う（Breg）ルーチン（等号"＝"の中間コード＄Ｅ６の有無をチェック）
- ＄９２Ｃ３→式の評価ルーチン………**３—８—２**節　参照
- ＄ＢＣＣ１→強制変換ルーチン（ＦＡＣ１をAregが示す型に変換する）
- ＄Ｂ３３５→「ＦＡＣ➡変数型」変換（変数代入）ルーチン…………**５—２—６**節　参照
- ＄０２８７→ディスクモード時の文字列代入用補助ルーチン
- ＄９７９０→Bregの長さ分だけ文字列領域を確保するルーチン
  ※この領域に対するＳＤが格納されるのが（７Ｅ～８０）である．
- ＄９８Ｅ４→Xregで示される番地からBregのバイト分の文字列を＄９７９０などで確保した領域に転送するルーチン
- ＄９９１Ｂ→Ｘ＝ＳＡＣポインタ（２１：２２）ならば，ＳＡＣポインタ（２１：２２）と（１Ｅ：１Ｆ）を－３してＳＡＣを一つ潰す（ＦＡＣのpull操作に相当する）．

＄９８５０→Ｘ番地の文字列が文字列スタック領域のものであれば，ゴミ化するルーチン

＄８５９７→ブロック転送（Ｘ＋→Ｕ＋；Bregのバイト分）

＄９８６５→Ｘ番地のＳＤに対する文字列リンクポインタを格納

解　説 代入文の処理は，**図３・８・１**に示すようなアルゴリズムで実行されます．文字列代入は，ＳＡＣなどに蓄積された，評価結果を表す**ＳＤ**を，変数本体に転送した後，文字列リンクポインタを修正する，といった手順になります．ただし，文字列スタック領域中の**文字列**と**ＳＤ**とは１対１に対応（ＳＤ⟺文字列リンク）していなければならないので，文字列を表す**ＳＤ**が変数エリア内のものであって，しかも，その**ＳＤ**の指す文字列が文字列スタック上のものである場合には，新たに文字列スタック上に領域を確保し（＄９７９０），文字列を**コピー**（＄９８Ｅ４転送ルーチン）するといった処理がなされています（＄９１ＢＡ～９１ＣＡ）．ＤＩＳＫモードのときは，もう少し複雑になり，上記以外に，**ファイルバッファ中の文字列**であった場合にも，文字列スタック領域に転送してから代入いたします．

図3・8・1　代入文(ＬＥＴ文)処理のブロック図

代入文　＄９１９５

変数サーチ(＄９５０Ｆ)

変数実効アドレスXreg→(５Ａ：５Ｂ)に格納

▼＝▼のチェック(＄Ｅ６→Breg ; ＪＳＲ　＄９２９４)

▼＝▼でない時 → Syntax Error

変数の型(１７)→Areg ;「ＰＳＨＳ␣Ａ」で，変数の型の退避

右辺の式の評価(＄９２Ｃ３)

「ＰＵＬＳ␣Ａ」で，変数の型，復帰

ＦＡＣを,Aregで示される型に強制変換(＄ＢＣＣ１)

＄９１ＡＤ…ＣＭＰＡ␣＃０３

数値型(≒＄０３)

ＬＢＮＥ　＄Ｂ３３５

数値代入(＄Ｂ３３５):ＦＡＣ→[(５Ａ:５Ｂ)~]　ＲＴＳ

文字型(＝＄０３)

ＳＤ位置(７６：７７)→Xreg ; 文字列実効アドレス→Dreg

ＲＯＭモード時

ＤＩＳＫモード時(０２８７→７ＦＣ４)

Dreg≦(３Ｆ：４０)
または
Xreg＜(３５：３６)

文字列の長さ(,Ｘ)→Breg

文字列格納用エリアの確保(＄９７９０)

エリアに対するＳＤの位置(６Ｄ：６Ｅ)→Xreg

確保したエリアに転送(文字列コピー)

Xreg＝＄００７Ｅ(一時的ＳＡＣの先頭)

ファイルバッファ先頭(７１Ｄ８：Ｄ９)≦Ｄ＜(３３：３４)
または
Ｄ＞(３Ｆ：４０)かつ
Ｘ≧(３５：３６)
すなわち，ファイルバッファ上の文字列も転送を行う

＄９１Ｄ０……………………

Dreg＜(７１Ｄ８：Ｄ９)or(３３:３４)≦Dreg

Dreg≦(３Ｆ：４０) or Xreg＜(３５：３６)

ＳＤの位置アドレス(Xreg)→(６Ｄ：６Ｅ)に退避

ＳＤがＳＡＣのものなら，ＳＡＣポインタをデクリメント(－３)する

Xreg←(５Ａ：５Ｂ)として，ゴミ化(＄９８５０)：古い変数データを消去

Breg＝３(バイト数)として，ブロック転送ルーチン＄８５９７により(６Ｄ：６Ｅ)の場所にあるＳＤを(５Ａ：５Ｂ)の変数本体に転送

＄９１Ｅ３……………………
ＪＭＰ　＄９８６５

＄９１Ｅ５

文字列リンクポインタの修正(＄９８６５)……………　ＲＴＳ

3—8—1　変数サーチ—変数の検索・登録などを行うルーチン—

アドレス　＄950F，12，14　（＄94CE　～　＄972B）

機　能　**＄950F**——最も一般的なエントリで，代入文・式の評価をはじめ，ほとんどの処理系がこのエントリを用いる．単純変数・配列変数の各要素などはこのエントリからの処理によって全て参照することができ，各要素の実効アドレス（変数のデータ部の先頭番地）を**Xレジスタ**および（**58：59**）に格納して戻る．

＄9512——**DIM文**用のエントリで，Bレジスタに0以外の値を入れてこの番地をコールすることによって，**DIM文実行中フラグ**（0016）が立ち，DIM文のパラメータとして書かれている配列の登録を行う．なお，DIM文では単純変数の登録をも行うことが可能であり，例えば「DIM␣A%，A%(9)，B！，B！（9）」などという文によって，A%(0)～A%(9)，B！(0)～B！(9)はもちろんのこと，A%やB！までも，初期値0として登録することができるようになっている．

＄9514——DEF␣FN文・FN関数処理系からのエントリです．これらの処理系では，**添字評価禁止フラグ**（1A）に＄80を入れて呼出してきます．

入力情報　**（001A）＝添字評価禁止フラグ**

- ＄00——普通の処理用で，変数名のすぐあとに左カッコを見つけると，**配列変数サーチ**＄95E6へ分岐する．
- ＄01——GET＠・PUT＠処理系からのオーダで，配列名のみ検索し，格納番地をXレジスタに入れてリターン．
- 上記以外—左カッコを見つけても配列とはみなさず，単純変数用の変数検索・登録を行う．

リターンアドレス（スタック最上位2バイト）：この値が＄92AFに等しい，即ち，**単項式の評価ルーチン**（3—8—3節）から呼出された場合には，未登録の単純変数名が出てくると，登録は行わずに，値**0**（文字列型なら長さ0）が書かれているエリアの先頭番地＄**B21B**をXレジスタおよび(58：59)に入れて戻る．

（D9：DA）＝読込みポインタ

変数名の1文字目の位置にあたる値を入れておく（1文字目が英大文字でない場合にはSyntax Error発生というしくみになっている）．

復帰情報　最も一般的には，**変数の実効アドレス**が**Xレジスタ**と（**58：59**）とに格納される．

(１A)＝１の場合，求める配列変数が登録されていれば，リンク部のアドレスをXレジスタに入れ，Aレジスタ＝0としてリターンし，未登録の場合には，Aレジスタ＝１としてリターンする．
(１６)≠0（DIM文）で配列の登録を行った場合には，復帰情報は特にない．
また，式の評価からのエントリで，未登録の単純変数であった場合には，Xレジスタに＄B２１B（前述）を入れて戻る．

解　説　変数サーチルーチンは，次のような各ブロックに分かれます．

(1)　＄９４CE～９４FC――変数名をテキストから読出して，**変数名参照用テーブル**（０５６８～０５７７）に積み込むルーチンです．

(2)　＄９４FD～９５０５――アルファベットチェックルーチン　｝前述

(3)　＄９５０６～９５０E――数字チェックルーチン　｝前述

(4)　＄９５０F　――変数サーチ・**メイン**エントリ

(5)　＄９５１８～９５４６――（１）を用いて変数名をロードした後，**型宣言文字**や**DEF型指定用テーブル**（０３１F～０３３８）を見て，**変数の型**を判断して（００１７）に格納するサブルーチンで「JMP␣＄D２」で呼出し元（＄９５１６）へ戻ります．

(6)　＄９５４７～９５６F――（８）を用いて単純変数の検索を行うブロックで，発見されれば，その**実効アドレス**をXと（５８：５A）に入れてリターン；発見されなければ，（７）へ進むことになります．
配列変数サーチなどへの分岐は，このブロックの頭部で行います．

(7)　＄９５７０～９５AE――未登録の単純変数を新たに登録するブロックです．ただし，スタックを調べて**式の評価**から呼出されていることを確認すると，登録は行わず，Xおよび（５８：５９）に＄B２１Bを格納してリターンします．登録が行われた場合には，その**データ部のアドレス**をXと（５８：５９）に入れて返します．

(8)　＄９５AF～９５C６――参照テーブル上の変数名と変数エリアのものとを比較していき，エリア上の１つの変数との一致不一致を検索するルーチンで，一致したら**ゼロフラグ**を立てて戻ります．

(9)　＄９５C７～９５D８――**変数の型**（００１７）と**変数名の長さ**から，**変数識別子**を作成するルーチンです．

(10)　＄９５D９～９５E５――配列変数サーチにおける，**添字の評価**を行うためのルーチンで，使用している主なサブルーチンは＄９２BE(式

の評価＋文字型エラーの型判断)，**＄ＢＣ７４**（整数型に変換）などで，評価結果が**負**の整数になった場合「Illegal Function Call」エラーを発生させます．

（１１）　**＄９５Ｅ６～９６６Ｄ——配列変数サーチ**を行うブロックです．
ＤＩＭ文中で，既に登録済の配列だった場合，「Duplicated Definition」エラーを発生させる一方，未登録の場合，**（１２）**へ進み登録を行います．**ＤＩＭ文でない時**，登録済で次元が一致した場合(一致しないと，「Subscript Out Of Range」になる）には，**（１３）**へ入り，要求された添字に対する要素の**実効アドレス**を割出す一方，未登録の場合には**（１２）**へ進んで**暗黙定義**（ＤＩＭ文によらない配列定義）による登録をした後**（１３）**へ入ります．

（１２）　**＄９６６Ｅ～９６ＢＢ**——配列変数を新たに登録するブロックで，**ＤＩＭ文の場合には**オーダされた**次数**(添字＋１の値)を格納しながら掛合わせていき，配列変数エリア内に格納領域を確保します．データ部を全部**＄００**で埋め，**リンク**を計算・格納してリターンします．**ＤＩＭ文以外の場合は**，暗黙定義時の添字**１０**に対する次数**＄０Ｂ**となるのが主な違いで，領域の初期化が終わるとリターンせずに**（１３）**に進んで，要求された添字に対する実効アドレスを計算します．

（１３）　**＄９６ＢＣ～９６Ｆ０**——配列の添字に対する**実効アドレス**を計算してＸレジスタと（５８：５９）とに格納するブロックです．例えば，「ＤＩＭ␣Ａ（d_1，d_2，d_3)」で宣言された配列変数Ａに対して，「Ａ（x_1，x_2，x_3)」を参照しようとした場合，実効アドレスは

〔Ａ（０，０，０）の位置〕＋**型**＊（x_1+d_1＊（x_2+d_2＊x_3））

なる式で与えられます．添字の順序に注意してください．

（１４）　**＄９６Ｆ１～９７２Ｂ**——**（１２）**や**（１３）**における乗算を実行するサブルーチンです．

これらの処理系によって，**２—２節**で述べたような構造の変数エリアが形成されていくわけです．

3—8—2　式の評価ルーチン

| アドレス | $92C3，92C9，92CB，92CD　～　$94C1

| 機　　能 | 代入文の右辺や関数の引数などに使われている式を評価して，その結果をFAC1に格納してリターンする（文字列式の場合，評価結果の文字列を表現するSD(一般的にSACである場合が多い)が置いてある番地の値をFAC1に入れて返す）．

$92C3——最も普通のエントリ

$92C9——PRINT文中のTAB(　)関数処理系などからのエントリ

$92CB ⎫ リカーシブコール（再帰的呼出）用のエントリポイント
$92CD ⎭ 〔呼出直前にピックアップした演算子の優先順位を入れて〕

| 入力情報 | $92CBエントリの場合，ワークレジスタ（005C）に直前の演算子の優先順位の値を格納して呼出す．

$92CDエントリの場合，Aレジスタに直前の演算子の優先順位を格納しておく．

| 復帰情報 | FAC1に，式の評価結果の数値；（0017）にその数値の型が格納されて終わる．

| WORK | FAC1（74～7C）と，その数値型（0017）

FAC2（82～8A）と，その数値型（005D）

（005C）＝式の優先順位一時格納用レジスタ

（0015）＝倍精度フラグ

（005E）＝比較演算子ピックアップ・参照用レジスタ

（008B）＝FAC1の符号 ⊕ FAC2の符号

（両FAC符号部の排他的論理和；5章参照）

（D9：DA）＝汎用読込ポインタ

（B6：B7）＝ポインタ（D9：DA）一時退避用

（7E～80）＝一時的SAC（文字列比較で用いる）

（23：24）＝リターンアドレス一時退避用

※（886C～888F）に演算子優先順位・エントリ・テーブルがある

| SUB | $D2ルーチン；$D8ルーチン

$8DAA→メモリフルテスト

$91ED→単項式の評価ルーチン（次節）

$974D→Dレジスタに形成された整数型データを最終評価

$98AE→文字列加算（SUBとして$98F4，F8を使う）

＄Ｂ２３６→除算実行ルーチン

＄Ｂ３Ｃ０→符号拡張ルーチン

＄ＢＡＦ７→ベキ乗（＾）処理

＄ＢＣ７４→整数化；＄ＢＣＣＤ→倍精度化；＄BCE0→単精度化

＄ＢＣＡＥ→（文字型エラーの）判断ルーチン

＄ＢＣＦＦ→整数型⇨単精度型・型変換ルーチン

※＄Ｂ２３６～＄ＢＣＦＦについては，第5章を御参照下さい．

終了条件 演算子が入るべき所に演算子以外の文字（例えば，文の区切り）があるとき「ＰＵＬＳ　ＰＣ，Ａ」でこのルーチンを抜け出す．また，リカーシブコール時，スタック上にプッシュしてある演算子優先順位が，後から読込んだものの順位以上である場合には，「ＰＵＬＳ　ＰＣ，Ａ」によって先の演算子に対する処理を行っていた式の評価ルーチンへとリターンする．

解　　説 **式の評価ルーチン**は，行番号定数などの若干の定数を除いた，あらゆる引数（コマンド，関数のパラメータ）を表現している式の評価を行います．第5章に述べてある各種の**数値演算処理系**や**文字列演算**などへの分岐を行う司令基地でもあります．

さて，数式中の**演算子**には**優先順位**があって，例えば，「乗除算（＊，／）は加減算（＋，－）よりも先に行う」などといった規則体系となっています．

前から順番に評価していくだけのことならば，単純なアルゴリズムで済みますが，**演算子の優先順位**を考慮して演算の後先を判断し，さらに，関数処理やカッコでくくられた項の処理までも行うとなると，非常に複雑なアルゴリズムが必要となります．これらの問題は，**リカーシブコール**（再帰的呼出；あるルーチンが自分自身をサブルーチンとして呼出すこと）と呼ばれる手法を用いることによって解決しています．リカーシブコールというのは，ＰＡＳＣＡＬなどの高級言語で好んで用いられる手法ですが，このようにインタプリタが中間言語を解釈して実行に移す上で必要不可欠なものです．それでは，**図３・８・２**に，式の評価ルーチンのアルゴリズムの流れを表すブロック図を掲げます．

優先順位の判定をスムーズに行うため，リカーシブコールの瞬間（呼出直後）に，先に読込んだ演算子の**優先順位を表す値**を**スタック上にプッシュ**しておきます．優先順位を表す値は，＄２８（▼ＩＭＰ▼）～＄７Ｆ（▼＾▼ベキ乗）の範囲にあり，後から読込んだ演算子に対する値と，スタック上の値とを比較することによって優先順位の判定を実現しています．もし，後からの値の方がより高位

図3・8・2　式の評価ルーチンのブロック図

※演算は◎で代表させてあります.
※「リカーシブ」と書いてあるのは「リカーシブコール(再帰的呼出)」の略です.

$92C3　ポインタ(D9：DA)デクリメント

$92C9　優先順位の区切$00をとる

$92CB
$92CD
リカーシブコール用エントリ

$92CD　優先順位をスタックにプッシュ

$92D4　単項式の評価($91ED)

$92D7　演算子サーチ

再び演算子をサーチ

$92E9　比較演算子？
yes → $9433 優先順位判定
(後)≦(先) → リターン
→ 比較演算
$9465 数値比較……四則演算 ($9344) をコール → $944A〜9457 符号拡張 ($B3C0) など
$943E $946B 文字列比較($946B)

no

$9302　2項演算子？
no → リターン
yes　(後)≦(先)

$9327　優先順位比較 → リターン
→ 文字列の"+" → $984E → 文字列最終評価($97AD)
(後)>(先)

$9329　FAC1型判断
$939E
→ ベキ乗"^" → FBC1 単精度化 ($BDE0) → FAC1 プッシュ → リカーシブ $92CB → スタックからFAC2にプル
$BAF7…… ベキ乗処理系

$9336　各処理系のエントリ読込

$93BB
整数除算 "¥","MOD"
$93B0
論理演算 "AND","OR"
$93BF FAC1 整数化 ($BC74) → $93C2 FAC1 プッシュ ($93E8) → $93C4 リカーシブ $92CB
$93BD→9384 FAC1⇔FAC2 交換(出現順に) → 整数除算系 FAC1◎FAC2
$93BF　FAC1整数化
$93B4　FAC1→Dreg ← スタックから第1引数をFAC2にプル
論理演算処理系(Dreg) FAC2◎Dreg → Dreg→FAC1 整数型宣言
$93B6の「JSR＿,X」によってコールする.　$974D

比較演算 ($9465) よりコール

$9344　四則(比較)演算 [+, −, *, ／; 比較]

$9344　FAC1(演算子の左側の評価結果)をスタックにプッシュ(93E8)

$9347　リカーシブコール($92CB)によって演算子の右側をFAC1に評価

$934A　FAC2にプル ($9411)
FAC2が倍精度型なら → $9353 FAC1倍精度化 倍精度フラグセット

$9359　FAC1の型判断
両方整数型なら → FAC1⇔FAC2

FAC1⇔FAC2

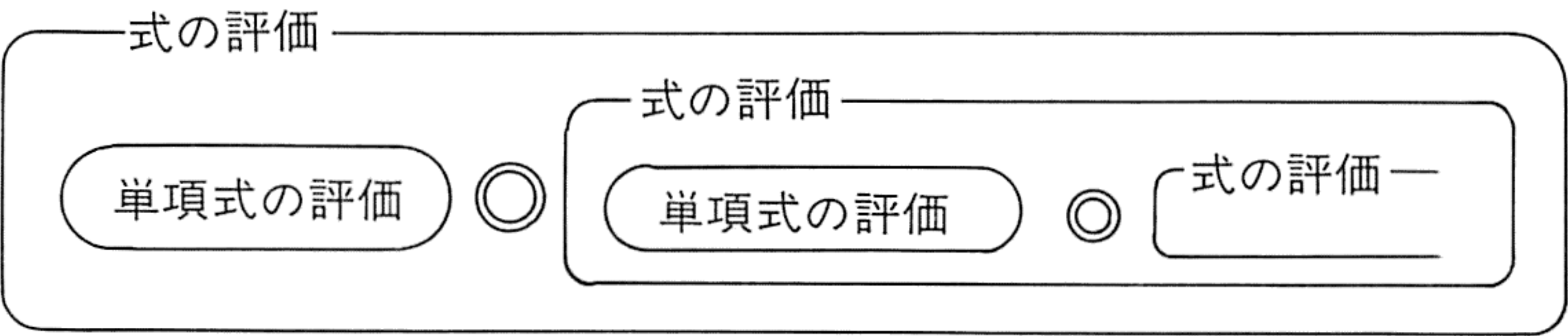
型合せ(精度の高い方に)

※双方整数型の場合, 一度しか交換を行わないため FAC1と2の順序が他のものと異なる.

$9358の「RTS」により分岐
四則演算・比較演算の各処理系
整数：　FAC1 ◎ FAC2
実数：　FAC2 ◎ FAC1
+：$BD4D　−：$BD3F
*：$BD73　／：$93CD
大小比較：$BE44
※四則：結果はFAC1に.
※除算(／)は最後$B236へ.

のものであった場合，さらにリカーシブコールを重ねて後の方の処理を優先して行う一方，先に読込んだ方が高位か等しい優先順位である場合にはリターンして先の演算子の処理を行います（このリターンは，スタックの位置が狂わないように，ＲＴＳ命令ではなく「ＰＵＬＳ　ＰＣ，Ａ」によって行われます）．従って，演算子が優先順位の低い順に並んでいるような式では，演算子の数だけリカーシブコールが重ねられることになります．なお，このルーチンの入口（＄９２Ｃ９）では，優先順位の区切として**＄００**を取込んでおり，リカーシブコール用の入口（**＄９２ＣＢ**と**＄９２ＣＤ**；両者の違いは，先に読込んだ演算子の優先順位の値を（**００５Ｃ**）に蓄えて入るか，**Ａレジスタ**内に置いて入るかの違いだけです）はこの後ろに配置されております．

式の評価ルーチンのサブルーチンのうち，**単項式の評価ルーチン**（＄９１ＥＤ）が特に大きく重要なもので，演算子をはさんでいる，**変数・関数・定数・カッコでくくられた式**などを評価してＦＡＣ１内に結果を格納する役割を担っております．式の評価ルーチンと単項式の評価ルーチンとの関係を模式的に表すと，次のようになります．ただし，２項演算子を▼◎▼で代表させます．

式の評価
単項式の評価　◎　式の評価
単項式の評価　◎　式の評価

単項式の評価と**リカーシブコール**とによって，演算子の両側のパラメータが出揃い，種々の前処理が終った後，各処理系へ分岐するわけですが，各処理系のエントリ番地は，（**８８６Ｃ～８８８Ｆ**）のテーブルに，**優先順位を表す値**とともに書かれており，その様子を**図３・８・３**に記しておきます．注意して欲しいことは，（８８７９：８８７Ａ）に書かれている**＄ＢＥ４４**は，ベキ乗▼＾▼処理系のエントリではなく，**比較演算処理系のエントリ**であることです．ベキ乗の場合，優先順位**＄７Ｆ**を見た瞬間に**ベキ乗前処理ブロック**（＄９３９Ｅ～＄９３ＡＦ）に入り，このブロックの最後に書かれている**＄ＢＡＦ７**へのＪＭＰ命令で**ベキ乗**処理系へと分岐するわけです．

比較演算子に関しては，少し説明を加えておかなければなりません．
比較演算子には，▼＜▼，▼＝▼，▼＞▼の３種類があり，重複しないかぎり，一時に３種類まで指定することができます．３種類のうちどれが指定されているかをピックアップする処理は**＄９２Ｅ１～＄９２Ｆ６**で行われ，その結果は，以下のように，ワークレジスタ（**００５Ｅ**）のビット０～３に格納されます（各演

図3・8・3，演算子の優先順位・処理系エントリ番地のテーブル

アドレス	優先順位	エントリ	
＄886C：	＄79	＄BD4D	（中間コード＄D9，▼＋▼）
6F：	79	BD3F	（　〃　＄DA，▼－▼）
72：	7C	BD73	（　〃　＄DB，▼＊▼）
75：	7C	93CD	（　〃　＄DC，▼／▼）
78：	7F	**＄BE44**	（　〃　＄DE，▼＾▼）
7B：	50	94A8	（　〃　＄DF，▼AND▼）
7E：	46	94AD	（　〃　＄E1，▼OR▼）
＄8881：	3C	94B2	（　〃　＄E2，▼XOR▼）
84：	32	94B7	（　〃　＄E3，▼EQV▼）
87：	28	94BC	（　〃　＄E4，▼IMP▼）
8A：	7A	BE33	（　〃　＄E5，▼MOD▼）
8D：	7B	BE0C	（　〃　＄E6，▼￥▼）

※1.　＄8878番地の＄7Fは，ベキ乗の優先順位であるが，
＄8879～A番地に書かれている＄BE44は，ベキ乗処理系のエントリではなく，**数値型の大小比較演算の処理系エントリ**なので注意して下さい．

※2.　優先順位を表す値は大きければ大きい程，優先度が高いわけで，
ベキ乗が最も優先度が高く，▼IMP▼が最も低くなっています．

※3.　＄886C～8Dのテーブルに載ってはいませんが，
比較演算子（▼＜▼，▼＝▼，▼＞▼）の優先順位は**＄64**，単項演算子▼－▼（負符号）の優先順位は**＄7D**，否定演算▼NOT▼は，＄5Aとなっています．

算子に対応するビットが１である時，その演算子に対する指定があったことを表します）．

図３・８・４　比較演算子の指定の仕方

						b_2	b_1	b_0
＄００５Ｃ	0	0	0	0	0	<	＝	>

比較演算処理系（数値型：**＄ＢＥ４４**；文字列型：**＄９４６Ｂ**）では，（先のパラメータ）＜（後のパラメータ）なら**＄０１**を，（先）＝（後）なら**＄００**を，（先）＞（後）なら**＄ＦＦ**を，それぞれＢレジスタに返してきます．この値に＄０１を足して右へ１ビットローテート（キャリー取込みがカギ）することによって，「＜」の時は[1|0|0]が，「＝」なら[0|1|0]が，「＞」なら[0|0|1]がそれぞれＢレジスタの下位３ビットに入り（上位５ビットは０），（００５Ｅ）とのＡＮＤをとることによって，指定された条件が成立するかどうかを知ることができる（０なら条件不成立）わけで，成立した場合には＄ＦＦをＢレジスタに読込んで，不成立の場合にはそのまま（既にＢレジスタ＝＄００となっている）**符号拡張ルーチン**＄Ｂ３Ｃ０を呼出すことにより，前者の場合には「－１」が，後者の場合には「０」がＦＡＣ１（整数型となるので（７６：７７）の２バイト）に格納されるわけです．

もう一つ説明しておきたいことがあります．それは，リカーシブコールの前後で，**ＦＡＣ１からスタック上へのプッシュ**（＄９３Ｅ８）および，**スタックからＦＡＣ２へのプル**（＄９４１１）の２つの操作が必ず加えられることです．
これらのサブルーチンを呼出すと，まず，リターンアドレスをスタックから読込んで（ＬＤＸ␣ ,Ｓ＋＋），さらに，ワークレジスタ（２３：２４）に退避させてしまいます．次に，前者の場合はＦＡＣ１の数値，その数値型の順に，**図３・８・５**のような形式でスタック上に積上げ，後者の場合はスタックからＦＡＣ２に読出します（読出したデータの数値型はワークレジスタ（００５Ｄ）に格納される）．
ＳＰの値を変えましたので，サブルーチンからのリターンは両者ともに「ＬＤＸ␣＄２３；ＪＭＰ ,Ｘ」で行います．

図3・8・5　ＦＡＣのプッシュとプル

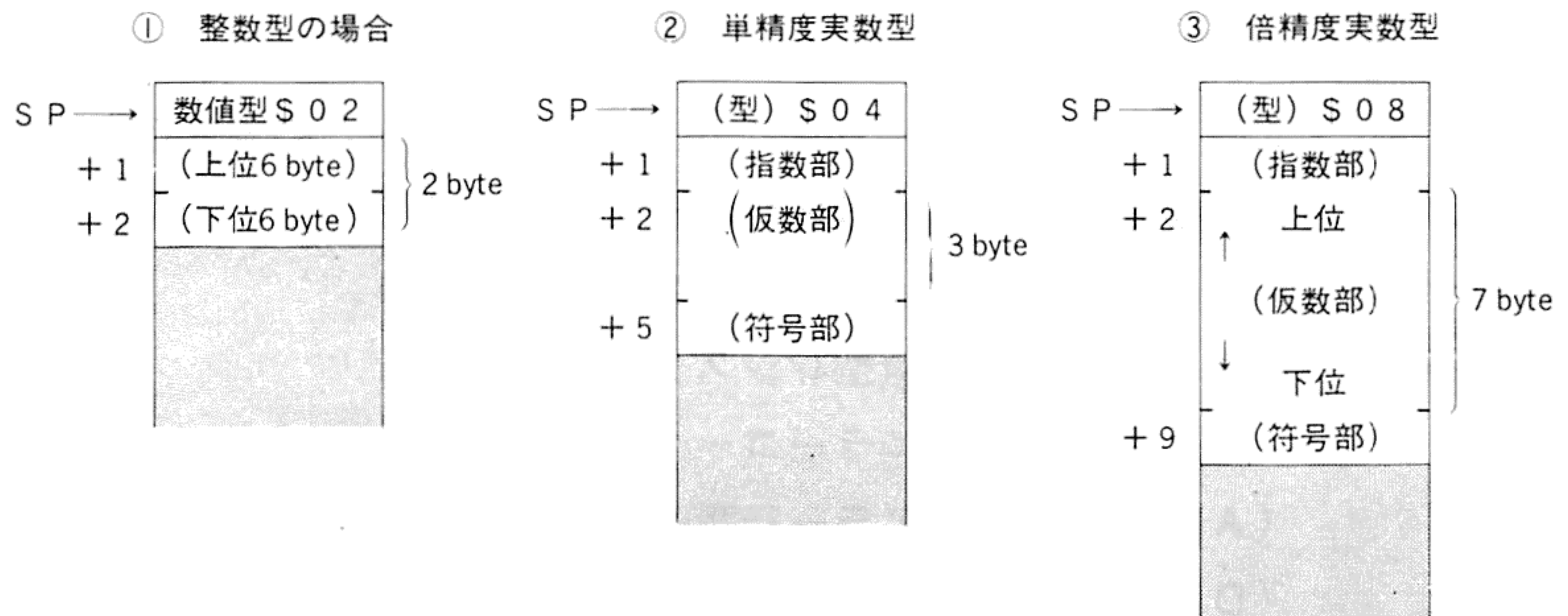
① 整数型の場合

ＳＰ→	数値型＄０２
＋１	(上位6 byte) 〕2 byte
＋２	(下位6 byte)

② 単精度実数型

ＳＰ→	(型) ＄０４
＋１	(指数部)
＋２	(仮数部) 〕3 byte
＋５	(符号部)

③ 倍精度実数型

ＳＰ→	(型) ＄０８
＋１	(指数部)
＋２	上位 ↑ (仮数部) ↓ 下位 〕7 byte
＋９	(符号部)

◀ 実数除算処理系　＄９３ＣＤ　～　＄９３Ｅ７ ▶

他の２項演算処理系でも同じことですが，この処理系にエントリする際には，ＦＡＣ１とＦＡＣ２の数値型を合わせておく必要があり，そのような処理は分岐前に通過した＄９３４４番地以下で既に実行済みです．

整数型で入ってきた時には，双方のＦＡＣの数値を単精度実数に変換（＄ＢＣＦＦ)するために「ＦＡＣ１⇔ＦＡＣ２」の交換を一度行いますので，演算順序は，「ＦＡＣ１／ＦＡＣ２」という具合に，実数の場合（ＦＡＣ２／ＦＡＣ１）とは逆になります（この処理系を単独で使用する場合には要注意)．

両ＦＡＣの符号部の排他的論理和をとって（００８Ｂ）に格納しておく点も重要です（第５章　参照)．この後，ＡレジスタにＦＡＣ１の符号を，ＢレジスタにＦＡＣ２の符号を読込んでから，除算実行ルーチン（＄Ｂ２３６）に入り，「ＦＡＣ２／ＦＡＣ１」を実行します．

◀ 論理演算ブロック　＄９４Ａ８　～　＄９４Ｃ１ ▶

論理演算子を▼◎▼で代表させると，「(ＦＡＣ２) ◎ (Ｄレジスタ)」の演算結果をＤレジスタに格納することになります．

◀ 整数型データ最終評価ルーチン　＄９７４Ｃ，＄９７４Ｄ▶

「ＣＬＲＡ；ＳＴＤ␣＄７６；ＬＤＡ␣＃２；ＳＴＡ␣＄１７；ＲＴＳ」といった手順の処理が行われます（後の３命令は＄ＢＣＡ０以下にある)．

3―8―3　単項式の評価ルーチン

| アドレス | $91ED　($91E8～92BD；$C5E2～C63C) |

| 機　能 | 変数・関数・定数・カッコでくくられた式などの実効値を評価して，

ＦＡＣ１内に結果を生成する．式の評価ルーチン（$92C3）と対の関係にあるルーチンであり，その関係は前節で述べたとおりである．

| WORK | ＦＡＣ１とその数値型レジスタ（0017）

（00AC）　＝エラーコード・レジスタ
（AD：AE）＝エラー行番号・レジスタ
（D9：DA）＝汎用読込ポインタ
（0272～）＝拡張フック（3バイト）
（0278～）＝　　〃
（027E～）＝　　〃

| SUB | ●外部ルーチン　（リカーシブコール：$92C3，CD）

$94FD→アルファベットチェック
$950F→変数サーチ
$974D→Dreg内の整数データの最終評価
$979B→文字列読出
$99BC→第２引数の評価用（MID$等の前処理用）
$99F4→ポインタ（D9：DA）の復帰（文字列処理用）
$9E7E→&定数の読出・評価
$A82A→ＦＮ関数処理系
$A9A0→ＵＳＲ関数処理系
$B301→（Xreg）番地の変数型数値をＦＡＣ１にロード
$BC74→ＣＩＮＴ（整数化；▼ＮＯＴ▼で使用）
$BCA5，AE→型判断（㊗数値型エラー，㊧文字列型エラー）
$BCE0→ＣＳＮＧ（単精度化；▼ＳＱＲ▼などの前処理）
$BD19→ＦＡＣ１内の符号無し整数（76：77）を単精度化
（行番号定数，▼ＥＲＬ▼，&定数などで使用）
$BDC9→ＦＡＣ１の符号反転（▼－▼で使用）

●汎用性の高い内部ルーチン

$9288→左カッコチェック＋式の評価＋右カッコチェック
$928C→右カッコチェック
$928F→左カッコチェック

＄９２９２→コンマチェック

＄９２９４→任意の文字コードチェック（Ｂレジスタ）

※以上のものは，様々な処理系から頻繁にコールされるので要注意．

解　説　このルーチンは，**式の評価**と**文字列加算**の２箇所からしか呼び出れませんが，巨大でしかも重要なものとなっています．そのアルゴリズムの概略は**図３・８・５**に示すとおりですが，巨大なわりには単純な流れであり，この図を見ればほとんど把握することができます．

さて，**＄９２９４～＄９２９Ｆでは，ポインタ（Ｄ９：ＤＡ）の指す位置にある文字コードがＢレジスタ内のコードに等しい**ことをチェックする（等しくなければ「Syntax Error」発生，即ち，＄９２Ａ０に分岐）処理を行っており，独立したサブルーチンとしても使用されますが，**カッコチェック**をはじめ，様々なバリエーションがあり，それぞれ使用頻度の高いものです．メモリ効率を上げるために工夫が施されていますので，参考のためにソースリストを掲げておきます．ユーザ・プログラムの中で，コンマチェックと式の評価または変数サーチを組合わせることによって，ＥＸＥＣ文に引数を持たせることが可能です．

＄９２８８～＄９２Ａ４のソースリスト

```
$9288  8D 05     BSR   $928F   左カッコチェック
       8D 37     BSR   $92C3   式の評価(リカーシブ)
$928C  C6 29     LDB   #$29    右カッコ▼)▼
       8C        CMPX  #……     ┐ 次の2バイトをスキ
$928F  C6 28     LDB   #$28    左カッコ▼(▼ ├ ップするためのダミ
       8C        CMPX  #……     ┘ ー命令
$9292  C6 2C     LDB   #$2C    コンマ▼,▼
$9294  9E D9     LDX   $D9     ポインタ読込
       A6 84     LDA   ,X      文字コード読取
       34 04     PSHS  B
       A1 E0     CMPA  ,S+     ┐ Bregに入っていたコードと等しいか？
       26 02     BNE   $92A0   ┘
       0E D2     JMP   $D2     ――ポインタ(D9:DA)をインクリメント
                                   してリターン
$92A0  C6 02     LDB   #$02    ┐ Syntax Error 発生
       7E 8DD1   JMP   $8DD1   ┘
```

図3・8・6　単項式の評価ルーチンのブロック図(2の1)

S91ED：単項式の評価エントリ；拡張フック（S0272）

S91F5　文の区切？ → yes → Missing Operand エラー（S91E8）

no

S91F7　SFE▼数値定数▼？ → yes →
- S9205　SF2：行番号定数 → テキスト→FAC1 → SBD19 単精度化
- S9212　S01：1byte整定数 → テキスト→FAC1 → 整数型宣言
- S921B　S02：2byte整定数 → テキスト→FAC1 → 整数型宣言
- S9225　実数型定数 → テキスト→FAC1 →（実数型の場合 SB301を使用）

no

S9231　変数名(アルファベット)であるか？ → yes → S92AC：変数読込ブロック
- 変数サーチ(950F) → 数値 →（X+）→FAC1（SB301）
- 文字列 → Xreg→(76：77)

※変数SDの番地→FAC1

no

S9236　SDA▼ー▼(負符号)か？ → yes → S92A5　負符号の優先順位 S7DをAregに持って → リカーシブ S92CD → SBD19　FAC1の符号を反転

no

S923A　SD9▼+▼(正符号)か？ → yes 振り出しへ

no

S923E　S22(引用符)か？ → yes → S9242 文字列定数読出(979B) → ポインタ復帰

no

S924A　SD7▼NOT▼？ → yes → ▼NOT▼の優先順位 S5AをAregに持って → リカーシブ S92CD → S924E 整数化(BC74) → COM␣S76 COM␣S77

no

S925A　S26▼&▼か？ → yes → 9E7E：&定数読出 → 単精度化(BD19)

no

S9260　SD4▼ERR▼か？ → yes → S9264 エラーコード(00AC)→Breg → SD2ルーチン → Aregクリア → 最終評価 S974D

no

S926C　SD3▼ERL▼か？ → yes → S9270 SD2ルーチン → エラー行番号(AD：AE)→(76：77) → 単精度化(SBD19)

no

S9279　SD2▼USR▼？ → yes → SA9A0 USR関数処理系 → RTS

no

SCF▼FN▼？ → yes → SA82A FN関数処理系 → RTS

S927F　no

図3・8・6　単項式の評価ルーチンのブロック図(2の2)

$9285
$FF(2バイトコード)?
no
$9288
左カッコ?
(928F)
no
Syntax Error
$928A
リカーシブ
$92C3
右カッコ?
JMP␣$D2
$C5E2:$FF80▼SGN▼～$FFB4▼LPOS▼の処理
拡張フック(027E～0280)
$D2ルーチン;A→B(次のコードがBregに)
ROM時　:$8077
DISK時:$7386
▼SGN▼～▼DATE▼
no
JMP␣[$0201]:拡張関数の処理
yes
JSR␣$9288
▼SGN▼～▼PEN▼
(1引数の関数)
左カッコチェック
式の評価
右カッコチェック
▼SQR▼
～
▼ATN▼
単精度化
(BCE0)
※拡張フック
(0278～A)
あり
各処理系コール
▼RND▼～▼DATE▼(引数なし)
▼INSTR▼～▼STRING▼
左カッコチェック
▼MID$▼/▼LEFT$▼/▼RIGHT$▼
第2引数読込;引数プッシュ
左カッコチェック
式の評価
コンマチェック
($9292)
最終評価:数値型→$BCAE;文字型→$97AD
$7386:
DISK関数か?
yes
▼DSKI$▼か?
yes
各処理系
コール
no
no
(式):$9288
JMP
[0215]
no
$8077:
▼LPOS▼か?
yes

各処理系のエントリ番地が書かれているジャンプテーブル

$8816:　B3BE　.　B472　.　B3D5　.....　E190
　　　　(SGN)　(INT)　(ABS)　　　　(DATE処理系)
$735D:　7C15　.　7C3D　.　7C40　.....　75A4
　　　　(DSKF)　(CVI)　(CVS)　　　　(DSKI$処理系)
$8075:　C947
　　　　(LPOS処理系)

第4章　入出力主要部

4－1　入出力とファイル

ＦＭ－７のコンピュータ本体（キーボードを除く）と，周辺装置（以後“デバイス”とも記します）とのやり取りは，ＣＲＴへの出力の一部を除けば，全て同じサブルーチン，同じ形式で扱われます．

ここで，周辺装置とは，キーボード，スクリーン，カセット，プリンタ，ディスク，ＲＳ－２３２Ｃなどを指し，各々“ＫＹＢＤ：”，“ＳＣＲＮ：”，“ＣＡＳ０：”，“ＬＰＴ０：”，“０：”～“３：”，“ＣＯＭ０：”～“ＣＯＭ４：”というデバイス名を持っています（文法書２．８参照）．

文法書の２．８節を見ればわかるように，ファイルディスクリプタを必要するＢＡＳＩＣのコマンド・ステートメントは，**ファイル**という概念で周辺装置を統一し，各々のデバイスに対しての考慮はなされていません．コマンド・ステートメントの処理系が，デバイス名を手がかりに，デバイスに応じたサブルーチンを呼び出すところで初めてデバイスが選択されます（一部のものは，デバイス名を見た点で考慮されるものもあります）．

このようにファイルを導入すると周辺装置の違いをあまり意識する必要がなくなるので（ただし，入力と出力で若干の違いが出てきます），入出力の大部分は，ファイルという形で扱われています（プログラムも当然プログラムファイルというファイルで扱われています）．そこで，この節では，ファイルとファイルディスクリプタを中心に入出力の形態を捉えていくことにします．

4－1－1　ファイルとシステムファイルディスクリプタ（ＳＦＤ）

ファイルとは，「入出力媒体上の論理的なデータ集合」（文法書　２．８）のことを指しますが，周辺装置とのデータのやり取りをするために設けられています．プログラムもデータもすべてファイルという形で捉えられ，ファイルディスクリプタで指定されたデバイスとの間で入出力を行います．キーボードやスクリーン，プリンタなどは，一見その性格が希薄なようですが，それらのデバイスとのやり取りも，やはりファイルとして扱うことができます．例えば，「ＳＡＶＥ“ＳＣＲＮ：”」という命令も可能です（「ＬＯＡＤ“ＫＹＢＤ：”」は不可能です）．

さて，ファイルの属性を記述するものとして**ファイルディスクリプタ（ＦＤ）**があります．ＢＡＳＩＣのコマンドやステートメントで用いたファイルディスクリプタは，デバイス名やオプションやファイル名を記述していますが，これをもとにＢＡＳＩＣインタプリタは，内部の処理系が使うためのファイルディスクリ

図4・1・1　ＳＦＤ（システムファイルディスクリプタ）の構成

ビット　7　6　5　4　3　2　1　0

ビット7 → ディスクかどうか
{ 0……ディスクでない
{ 1……ディスクである

ビット6〜4 → Ｉ／Ｏモード

モード＼ビット	6	5	4
インプット	0	0	1
アウトプット	0	1	0
ランダム	1	0	0

ビット3〜0 → デバイスまたはファイルバッファ

●ディスクでない場合(ビット７＝０)
デバイス番号　＄０　ＫＹＢＤ(コンソール)
＄１　ＳＣＲＮ
＄２　ＣＡＳ０
＄３　ＬＰＴ０
＄４　ＣＯＭ０
〜
＄８　ＣＯＭ４

●ディスクの場合（ビット７＝１）
ファイルバッファ番号
＃＃０
〜
＃＃１５

図4・1・2　ＳＦＤ（システムファイルディスクリプタ）テーブル

(０２Ｂ４:０２Ｂ５)
ＳＦＤの先頭アドレス
(＝＄０７０Ｃ)
＋
(００ＢＦ)
ファイル番号
⇩
求めるＳＦＤの
アドレス

ファイル番号		アドレス
＃０	ＳＦＤ	＄０７０Ｃ
＃１	ＳＦＤ	＄０７０Ｄ
＃２	ＳＦＤ	＄０７０Ｅ
＃３	ＳＦＤ	＄０７０Ｆ
＃４	ＳＦＤ	＄０７１０
⋮		
＃１４	ＳＦＤ	＄０７１Ａ
＃１５	ＳＦＤ	＄０７１Ｂ
＃１６	ＳＦＤ	＄０７１Ｃ
＃１７	ＳＦＤ	＄０７１Ｄ

＄０７０Ｃ←標準用：コンソールへの入出力として使用され，値は＄００が定常的に入っている．

＄０７０Ｄ〜＄０７１Ｃ：ユーザ用のファイルに当てられる．ＯＰＥＮされていないときは，＄００が設定されている．

＄０７１Ｄ←システム用：プログラムの入出力やＬＰＲＩＮＴなどで使われる．

プタをワークエリア上に設定します．

このため，前者のファイルディスクリプタと後者のファイルディスクリプタとを，異なるものとして識別する必要があります．そこで，本書では，後者のＢＡＳＩＣインタプリタで使うファイルディスクリプタを**システムファイルディスクリプタ**と呼び**ＳＦＤ**と略します．

ＳＦＤは，そのファイルがアクセスしている周辺装置が何であるかを記述するものであり，また，ディスクをアクセスしているときには，どのファイルバッファ（後述）を使用しているかを示します．加えて，そのファイルが，入力として使われているのか，出力として使われているのか，ディスクの場合はさらに，ランダムファイルとしてそのファイルバッファを使っているかどうかを示します．従って，ＳＦＤより，そのファイルがどのようなオープン状態であるか，あるいは，そのファイルがオープンしているかどうかの判断（オープンしていないときは，＄００となっています）の基準にもなります．

１つのＳＦＤの内容は，**図４・１・１**のようなビット構成をとっており，そのビットの構成が前述した内容を持っているわけです．これは，デバイス番号（０２Ｅ６）とファイルモード（０２ＤＡ）のＯＲをとって作られます．

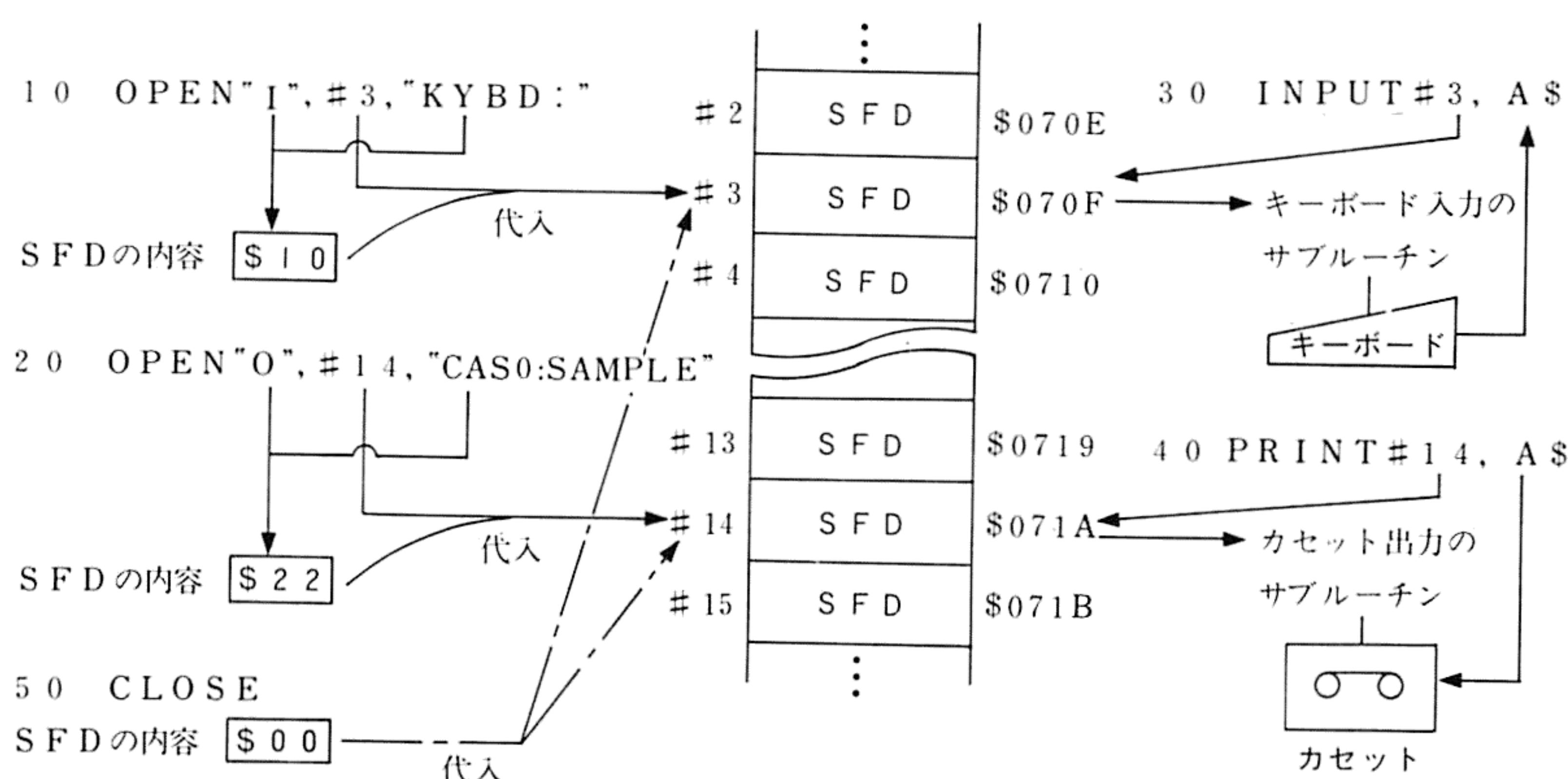
さて，ファイル番号が１から１６まで使用可能であるということから判るように，ユーザに対して，ファイルは１６個設けられています．そして，その分だけＳＦＤが１６個設けられています．その他に2個，**システム用**および**標準用**がありますが，標準用は，**コンソール**入出力（すなわちスクリーンに出力し，キーボードから入力するという意味）用で，常に＄００が入っています．

この計１８個のＳＦＤのテーブルは，（０７０Ｃ～０７１Ｄ）の領域にあり，その先頭アドレス＄０７０Ｃは，ワークエリア（０２Ｂ４：０２Ｂ５）に定数として格納されています．１つのＳＦＤをアクセスするときは，このワークエリアの値にファイル番号を足してそのアドレスを求めます．ＢＡＳＩＣインタプリタでは，ファイル番号レジスタとしてワークエリアに（００ＢＦ）を設けていて，（０２Ｂ４：０２Ｂ５）の値と，この（００ＢＦ）の値を足して，ＳＦＤ（システムファイルディスクリプタ）のアドレスを求めています（**図４・１・２**）．

実際にＢＡＳＩＣ上での動きを見ることにします．例えば，例１のプログラムはキーボードから入力した文字を，カセットへ出力するプログラムですが，実行中のＳＦＤは，１０行の「ＯＰＥＮ」で，ＳＦＤ＃３（＝＄０７０Ｆ）にキーボード入力として**＄１０**を，２０行の「ＯＰＥＮ」では，ＳＦＤ＃１４（＝＄０７１Ａ）にカセット出力として**＄２２**を，各々設定します．次の３０行の「ＩＮＰＵＴ」では，ＳＦＤ＃３の内容を見て，キーボードからの入力だということを知

り，キーボード入力のサブルーチンを呼びます．同様に４０行でもＳＦＤ＃１４の内容を見て，カセット出力のサブルーチンを呼びます．このように，ＳＦＤを参照してデータの入出力が行われます．なお，５０行の「ＣＬＯＳＥ」は，ＳＦＤの内容を＄００に戻します（図４・１・３）．

例１
```
10 OPEN"I",#3 ,"KYBD:"
20 OPEN"O",#14,"CAS0:SAMPLE"
30  INPUT#3 ,A$
40  PRINT#14,A$
50 CLOSE
```

図４・１・３　ＳＦＤ（システムファイルディスクリプタ）の使われ方

４－１－２　入出力のバッファ

バッファは，周辺装置への入出力データを一時的に保管する領域のことを指しますが，Ｆ－ＢＡＳＩＣでは，次のものが用意されています．

●**システム入力用　（０４３Ｄ～０５３Ｃ）**

周辺装置から入力したデータは最終的にこの領域に移され，この領域を利用して変数の代入を行い，また，テキストの作成を行います（これらについては，２章～３章参照）．

●**カセット用　（０５ＥＤ～０６ＥＢ）**

カセットとの入出力のためにに設けられているバッファですが，「ＰＬＡＹ」でもこのバッファ領域を使用しているため，カセットと「ＰＬＡＹ」を同時に使用することはできません．

●ＲＳ－２３２Ｃ用（０７８Ｆ～）

ＲＳ－２３２Ｃの個数により，大きさが変化します．また，各々のバッファの先頭部分には，入出力のサブルーチンへのジャンプテーブルが配置されています（１－３節参照）．

●ディスク用（ドライブテーブルとファイルバッファ）

ディスクのためのバッファは，ＲＳ－２３２Ｃが入ることによってその先頭アドレスが変わってくるため，また，ドライブの数とファイルバッファの数によってその大きさが変化するため，メモリのどこに存在するかは確定できません（これについては，１－４節に説明してあります．また，**表１・４・１**および**表１・４・２**には計算されたアドレスが求められています．次節も参照して下さい）．

●汎用ディスクバッファ（７２１３～７３１２）

上記のものは，テキストエリアの前にあるバッファですが，ＤＩＳＫモードのときは，ドライブテーブルとファイルバッファに加えて，汎用のディスク用バッファが取られます．その大きさは，１つのセクタとの入出力を行うため，２５６バイトになっています．バッファが使用される場合を次に列記します．

・ＦＡＴをドライブテーブルに入力するとき，あるいは，ドライブテーブルの内容をＦＡＴに出力するとき（これについては次節で述べますが，ＦＡＴは１セクタですから，２５６バイトであり，ドライブテーブルは１５８バイトの大きさですから，入出力の際，このバッファを利用して調整をはかっています）．

・ディレクトリ（ファイル名，ファイル形式，ファイルのクラスタ番号を記憶しているセクタ）の入出力および，その変更のとき．

・ＩＤセクタを読み込み，ＩＤのチェックを行うとき．

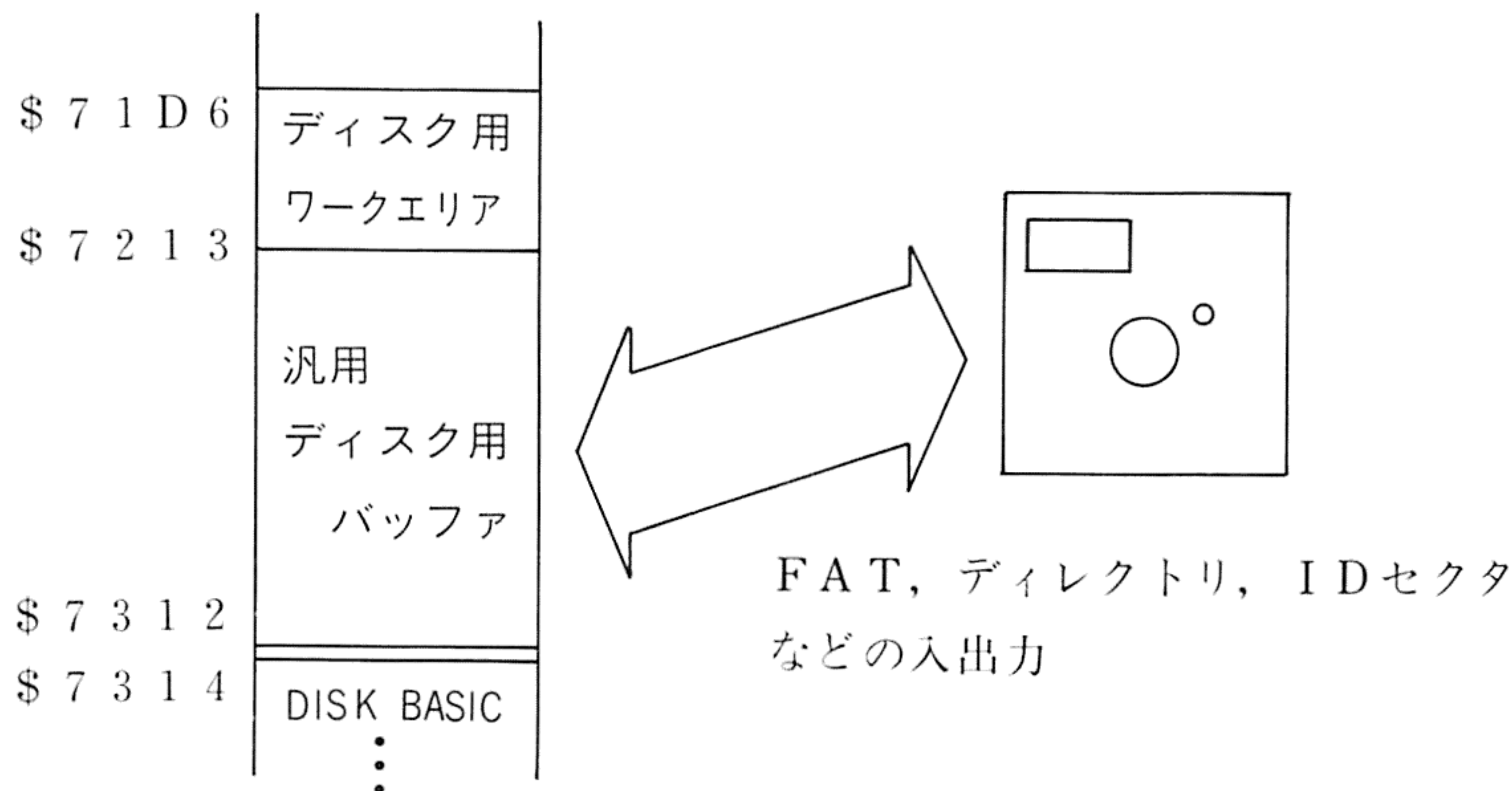
また，このバッファの前には，ディスク用のワークエリアが＄７１Ｄ６～＄７２１２の間に取られています（一部のものについては，１－４節で述べましたが，これらのものについてはディスクの節で述べることにします）．

＄０４３Ｄ
システム入力用バッファ
＄０５３Ｃ
＄０５ＥＤ
カセット用バッファ
＄０６ＥＢ
＄０７８Ｆ
RS-232C用バッファ
ドライブテーブル
ファイルバッファ
テキストエリア
そのデバイスがない場合は，作られない．

図４・１・４　バッファ

図４・１・５　汎用ディスク用バッファ

$７１Ｄ６　ディスク用ワークエリア
$７２１３　汎用ディスク用バッファ
$７３１２
$７３１４　DISK BASIC

ＦＡＴ，ディレクトリ，ＩＤセクタなどの入出力

４－１－３　ドライブテーブル（ＤＲＶＴＢＬ）

ドライブテーブルというのは，ディスク用のバッファの一種で，ディスクより**ＦＡＴ**（File Allocation Table）を読み込み，この領域で各クラスタの使用状況の更新を行い，再びＦＡＴに書くために用いられています（ＦＡＴは，トラック１のセクタ１にあります．文法書２．１０．３参照）．

ドライブテーブルの内容は，**図４・１・６**に示してあります．ドライブテーブルの第０バイト目は，そのドライブテーブルをアクセス（呼出し）しているファイルの数を持っています．この第０バイト目の値が＄００のときには，そのドライブテーブルをアクセスしているファイルがないことを示していますから，ディスクの入出力をする際に，新たにディスクのＦＡＴを，そのドライブテーブルに読み込んできます．また逆に，第０バイト目の値が＄００でないときには，その値だけ，そのドライブテーブルをアクセスしているファイルがあることを示しています．従って，ＦＡＴをディスクから読み込んであるので，新たにＦＡＴを読み込むということは行いません．即ち，第０バイト目は，**ＦＡＴを入力するためのパラメータ**としても使われているのです．

次の第１バイト目は，変更回数を示します．すなわち，ドライブテーブルの内容を何回変更したかを示しているのですが，変更回数がワークエリアの（０５Ｅ５）（＝記入レベル）の値以上になると，ＦＡＴへ書き込みます．ここで，（０５Ｅ５）には＄０１が設定されていますから，結局，１回ドライブテーブルの内容を変更するごとにＦＡＴへ出力することになります．出力した後は，第１バイト目の値は＄００に戻されますから，このバイトの取り得る値は，＄００（出力済）

と＄０１（出力要求）であると考えることができます．従って，このバイトは**ＦＡＴを出力するためのパラメータ**としても使われていることがわかります．また，第１バイト～第１５７バイトの間がＦＡＴに出力されますが，出力の際，第１バイト目は＄００に設定されています．

第２バイト～第５バイトは，未使用です．第６バイト～第１５７バイトは，クラスタの０～１５１に対応して，各バイトの意味はＦＡＴと同様です（文法書２．１０．３参照）．

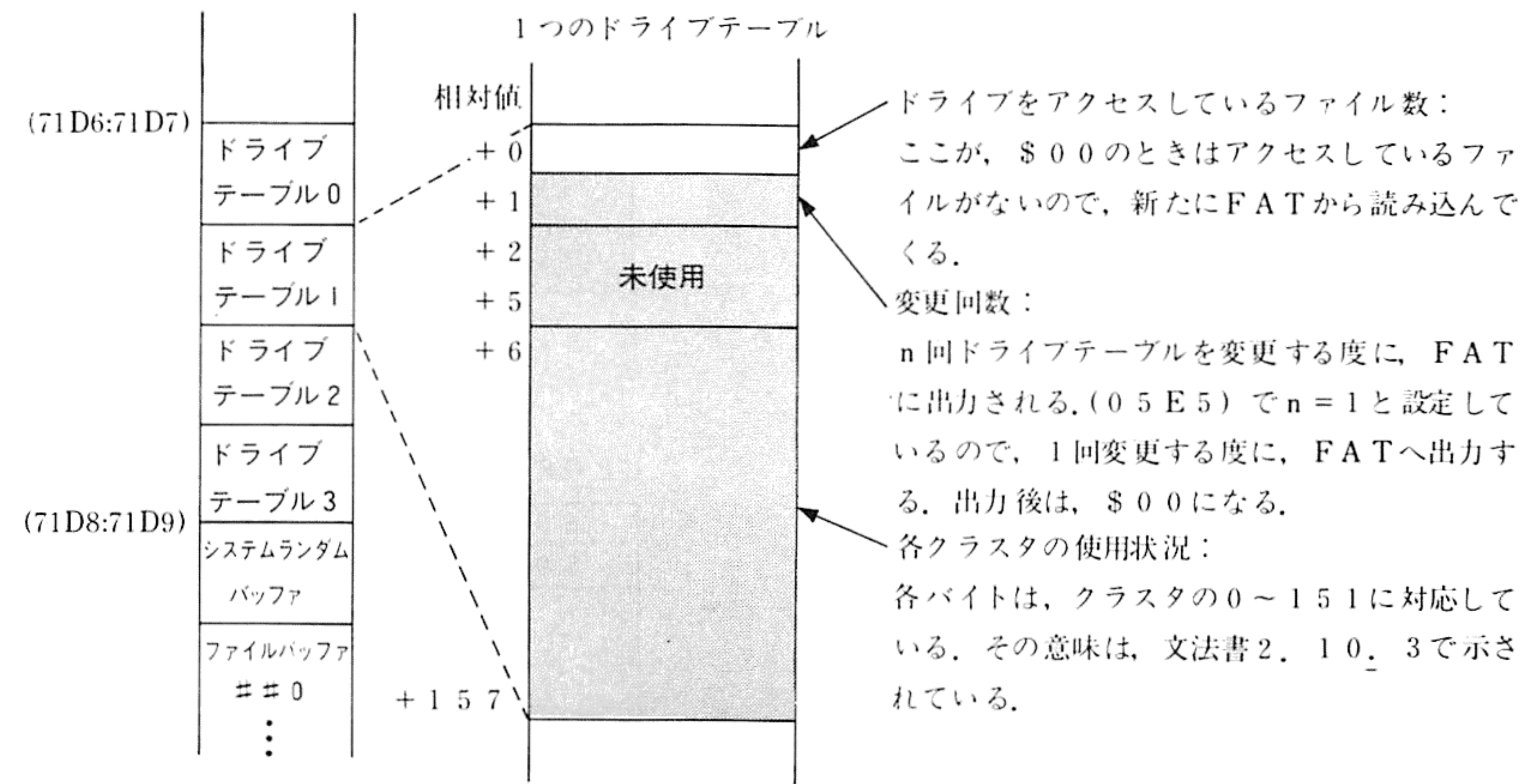
図４・１・６　　ドライブテーブルの内容

(71D6:71D7)
ドライブテーブル０
ドライブテーブル１
ドライブテーブル２
ドライブテーブル３
(71D8:71D9)
システムランダムバッファ
ファイルバッファ＃＃０
⋮

１つのドライブテーブル
相対値
＋０
＋１
＋２ 未使用
＋５
＋６
＋１５７

ドライブをアクセスしているファイル数：
ここが，＄００のときはアクセスしているファイルがないので，新たにＦＡＴから読み込んでくる．

変更回数：
n回ドライブテーブルを変更する度に，ＦＡＴに出力される．(０５Ｅ５）でn＝１と設定しているので，１回変更する度に，ＦＡＴへ出力する．出力後は，＄００になる．

各クラスタの使用状況：
各バイトは，クラスタの０～１５１に対応している．その意味は，文法書２．１０．３で示されている．

※　図に示したように，ドライブテーブルは最高４個とれるが，それだけのドライブが接続されていない場合は，ファイルバッファなどが順次，後ろから前に詰められていく．

※※　ＦＡＴに入出力するのは色のついた部分である．＋１バイト目は出力される際には，＄００の値をとる．

ドライブテーブルの変更をする場合，即ち，最終的にＦＡＴの変更をする場合には，次のものがあります．

- 「ＯＰＥＮ“Ｏ”～」または「ＯＰＥＮ“Ｒ”～」，「ＳＡＶＥ」または「ＳＡＶＥＭ」などで，新しいディスク上のファイルを作る場合．
- 「ＰＲＩＮＴ＃～」などで，データがそのセクタに収まりきらなくて，新たなセクタを必要とする場合．
- 「ＰＵＴ＃～」などで，新たなレコード番号を指定したため，新たなセクタを必要とする場合．
- 「ＫＩＬＬ」などで，既存のファイルを削除するときや「ＤＳＫＩＮＩ」でディスクの内容を初期化する場合．

以上の場合の例として，例2のプログラムを実行させたときの，ドライブテーブルの推移を図4・1・7に示します．

例2
```
10 OPEN "I",#2,"0:SAMPLE1" : OPEN"O",#5,"0:SAMPLE2"
20 WHILE NOT EOF(2)
30   INPUT #2,A$ : PRINT #5,A$ :PRINT A$
40 WEND
50 CLOSE #2 : CLOSE #5
```

図4・1・7　ドライブテーブルの推移

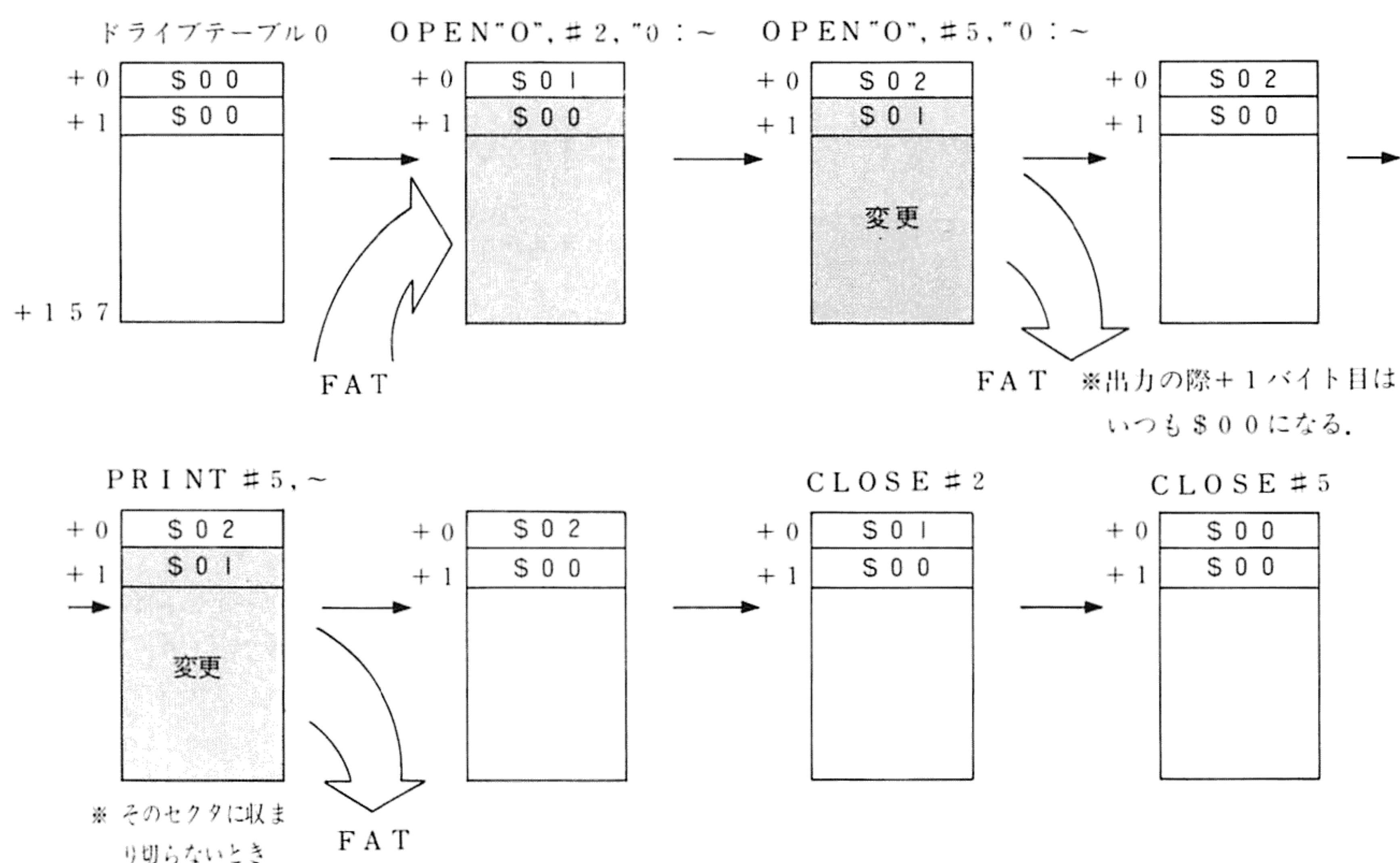
ドライブテーブル0　OPEN"O",#2,"0：～　OPEN"O",#5,"0：～

+0 $00　+1 $00　+157　→　+0 $01　+1 $00　→　+0 $02　+1 $01　変更　→　+0 $02　+1 $00　→

FAT　　FAT　※出力の際+1バイト目はいつも$00になる．

PRINT #5,～　CLOSE #2　CLOSE #5

+0 $02　+1 $01　変更　→　+0 $02　+1 $00　→　+0 $01　+1 $00　→　+0 $00　+1 $00

※そのセクタに収まり切らないとき　FAT

4－1－4　ファイルバッファ（FBUF）

ディスクに割り当てられたファイルのために，ファイルバッファが設けられています．1－4節で述べたように，ファイルバッファのうちシステムランダムバッファは「FIELD＃0～」用に，ファイルバッファ＃＃0はシステム用（プログラムの入出力と「DSKINI」）に用いられます．他の＃＃1～＃＃15のファイルバッファは，データファイルとして用いられます．ただし，BASIC起動時に設定したファイルバッファ数より多くのファイルバッファをオープンすることはできません．また，ユーザは，ファイルをオープンする際にファイル番号を指定しますが，ファイル番号はSFDの番号を指すもので，ファイルバッファ番号を指すものではありません（ファイルバッファ番号は，そのファイル番

号に対応するＳＦＤが持っています）．例えば，ユーザがファイル番号＃９でオープンしても，実際に使用されているファイルバッファは＃＃３となる場合があります．このようにファイル番号を論理的に与えることにより，少ないバッファを効率良く使用することができるのです．

１つのファイルバッファは**ＦＣＢ（ファイルコントロールブロック）**と呼ばれる部分と**Ｉ／Ｏバッファ**と呼ばれる部分から成り立ちます．このうち，Ｉ／Ｏバッファというのは，実際にファイルの内容が入る部分で，ディスクの１つのセクタに書き込むまたは読み込む内容が格納されています．ＦＣＢの部分には，そのファイルの属性が格納されています．具体的には，ファイルモードやドライブ番号やクラスタ番号などが書かれています．これについては，**図４・１・８**にその詳

図４・１・８　　　ファイルバッファ

(71D6:71D7)
ドライブテーブル０
ドライブテーブル１
(71D8:71D9)
システムランダムバッファ
(71DA:71DB)
ファイルバッファ＃＃０
ファイルバッファ＃＃１
ファイルバッファ＃＃２　271バイト
ファイルバッファ＃＃ｎ－１
ファイルバッファ＃＃ｎ
テキストエリア

※ｎは最大15まで

１つのファイルバッファ
ＦＣＢ（15バイト）
Ｉ／Ｏバッファ（256バイト）

ＦＣＢ（ファイルコントロールブロック）の内容

相対値	機能・解説	
＋０	ファイルモード；アクセスフラグ ＄００……未使用 ＄１０……インプットモード ＄２０……アウトプットモード ＄４０……ランダムモード	
＋１	ドライブ番号　（０～３） ファイルが使用しているドライブ番号	
＋２	ファイルの先頭クラスタ番号 （０～１５１〈＄９７〉）	
＋３	アクセス中（使用中）のクラスタ番号 （０～１５１〈＄９７〉）	
＋４	アクセス中（使用中）のクラスタ内のセクタ番号　（０～７）	
＋５	インプットモード時のＩ／Ｏバッファ中のデータのポインタ（０～＄ＦＦ）	ランダムモードのときは、両方でレコード番号を示す
＋６	アウトプットモード時のＩ／Ｏバッファ中のデータのレコード長（＄０１～＄ＦＦ）	
＋７ ＋８	Ｉ／Ｏバッファ長 ＄０１００	
＋９ ＋10	Ｉ／Ｏバッファの先頭アドレス	
＋11	１つのシーケンシャルレコードが，途中で切れたとき＝＄ＦＦ	
＋12	１つのシーケンシャルレコードが，途中で切れたときのデータの一時的保管場所として用いられる	
＋13 ＋14	Ｉ／Ｏバッファ中のデータのポインタ （＄００００～＄０１００）	

細を示しておきました．注意すべき点は，（ＦＢＵＦ＋５）のインプットモードのときのデータのポインタと，（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）のデータのポインタです．インプットモード時は，（ＦＢＵＦ＋５）はアクセスすべきデータのポインタであり，（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）は残りのデータの長さを示します．従って入力ルーチンは，１バイトのデータを持ってくるときには（ＦＢＵＦ＋５）を，Ｉ／Ｏバッファにデータがなくなったことを知るときには（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）を参照します．４－５節では（ＦＢＵＦ＋５）を**データポインタ**，（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）を**ポインタ**と呼び区別しています．※ＦＢＵＦはファイルバッファの先頭アドレスのことも示しています．

４－１－５　入出力のジャンプテーブル

基本的な入出力ルーチンは，デバイスの違いを考えずに入出力が行えますが，それは，ＳＦＤが設定されていることによります．本節では基本的な入出力ルーチンが，どのようにＳＦＤから各デバイスとの入出力を行うかについて述べます．

基本的入出力ルーチンでは，各デバイスごとの入出力ルーチンをサブルーチンとして呼び出しています．従って，実際には各デバイスの入出力ルーチンがそのデバイスとの入出力を行っています．

各デバイスの入出力ルーチンを呼び出すために，基本的入出力ルーチンは次のような方法を用いています．

（１）　そのファイル番号のＳＦＤを見る．

　（ｉ）　そのＳＦＤのビット７が立っていれば，ディスク入出力ルーチンを呼び出す．

　（ⅱ）　そのＳＦＤのビット７が立っていなければ⑵以下の動作をする．

（２）　そのＳＦＤの下位４ビットが示すデバイス番号を参照し，デバイス分類テーブルと呼ぶジャンプテーブルの先頭アドレスが格納されているテーブルから，そのデバイス番号のアドレスを得る．

（３）　⑵で求めたジャンプテーブルの先頭アドレスと，基本的入出力ルーチンが行おうとしている内容とから，ジャンプテーブル内のデバイス別入出力ルーチンのアドレスを見つけ，その入出力ルーチンを呼び出す．

デバイス分類テーブルは**図４・１・９**に，ジャンプテーブルは**図４・１・１０**に，ディスクのジャンプ先は**図４・１・１１**に示しました．それでは，実例を見ることにします．

図4・1・9　デバイス分類テーブル

ワークエリア	06EC:06ED	06EE:06EF	06F0:06F1	06F2:06F3	06F4:06F5
相　対　値	+0	+2	+4	+6	+8
デバイス名	KYBD(コンソール)	SCRN	CASO	LPTO	(COMO)
デバイス番号	0	1	2	3	4
アドレス	$DB0D	$D980	$C9E8	$0778	$078F

ワークエリア	06F6:06F7	06F8:06F9	06FA:06FB	06FC:06FD
相　対　値	+10	+12	+14	+16
デバイス名	(COM1)	(COM2)	(COM3)	(COM4)
デバイス番号	5	6	7	8
アドレス	$082E	$08CD	$096C	$0A0B

(02B2:02B3)
=$06EC

図4・1・10　入出力のジャンプテーブル

先頭アドレス　ⓐ	$00~$03	$04, $05	$06, $07	$08, $09	$0A, $0B	$0C, $0D	$0E, $0F	$10, $11
ⓑ	デバイス名	オープン	クローズ	1バイト入力	1バイト出力	ポジションチェック	EOF	LOF
ⓒ	$CC37	$CEDC	$CE4B	$D072	$D08E	$D0CA	$D11A	$D11D
$DB0D	KYBD	$DB1F	$DB25	$DB54	$D9D9	$D992	$CEF4	$CEF4
$D980	SCRN	$D99E	$D9A4	$DB54	$D9D9	$D992	$CEF4	$CEF4
$C9E8	CASO	$CA06	$CAB6	$CB02	$CB1E	$CB4D	$C9FA	$CA02
$0778	LPTO	$D9A5	$D615	$CEF4	$D617	$D39C	$CEF4	$CEF4
$078F	(COMO)	$D1AD	$D198	$D2B5	$D25C	$D39C	$D3A5	$D3B4
$082E	(COM1)	//	//	//	//	//	//	//
$08CD	(COM2)	//	//	//	//	//	//	//
$096C	(COM3)	//	//	//	//	//	//	//
$0A0B	(COM4)	//	//	//	//	//	//	//

ⓐ…先頭アドレスからの相対値
ⓑ…テーブルの内容
ⓒ…呼び出しているルーチン（エントリが複数あるものは，代表して1つ）

図4・1・11　ディスクのジャンプア

ワークエリアの先頭アドレス	(023C)	(023F)	(0245)	(0242)	(0248)	(024B)	(024B)
ジャンプ先の各ルーチンの内容	オープン	クローズ	1バイト入力	1バイト出力	ポジションチェック	EOF	LOF
呼び出しているサブルーチン	$CEDC	$CE4B	$D072	$D08E	$D0CA	$D11A	$D11D
ジャンプ先	$7713	$78BE	$79F9	$7924	$7AA8	$7CA4	$7CA4

なお，ＳＦＤが未設定の状態でジャンプテーブルを参照しなければならないオープン処理系などへの分岐の際は，デバイス番号レジスタ（０２Ｅ６）を参照することによって処理を進めています（**図4・3・2参照**）.

例3　カセットから1バイト入力する場合　図4・1・12（ファイル番号は＃3，ＳＦＤの内容は＄１２とする）

（1）　基本的入出力ルーチンの１つである１バイト入力ルーチン（＄Ｄ０７２）は，ファイル番号＃3のＳＦＤの内容を見ます（Ｂレジスタに入る）.

ＳＦＤの内容は＄１２であり，ビット７が立っていないのでディスクではないことがわかります（Ｎフラグを見ている）.

（2）　デバイス番号は，下位４ビットから２（＄０２）であることがわかり，デバイス分類テーブルからデバイス番号２のアドレス，すなわち＄Ｃ９Ｅ８を持ってきます（Ｘレジスタに入る）.

（3）　＄Ｃ９Ｅ８から始まるＣＡＳ０のジャンプテーブルのうち１バイト入力のものは＄ＣＢ０２であるから，サブルーチン＄ＣＢ０２を呼び出し，１バイトのデータを受け取ります（Ａレジスタに入る）.

図4・1・12　　各入出力ルーチンを呼び出す方法

ワークエリア
（０２Ｂ２：０２Ｂ３）＝デバイス分類テーブルの先頭アドレス（＝＄０６ＥＣ）
（０２Ｂ４：０２Ｂ５）＝ＳＦＤテーブルの先頭アドレス（＝＄０７０Ｃ）
（００ＢＦ）　　　　　＝ファイル番号（この例では＄０３）

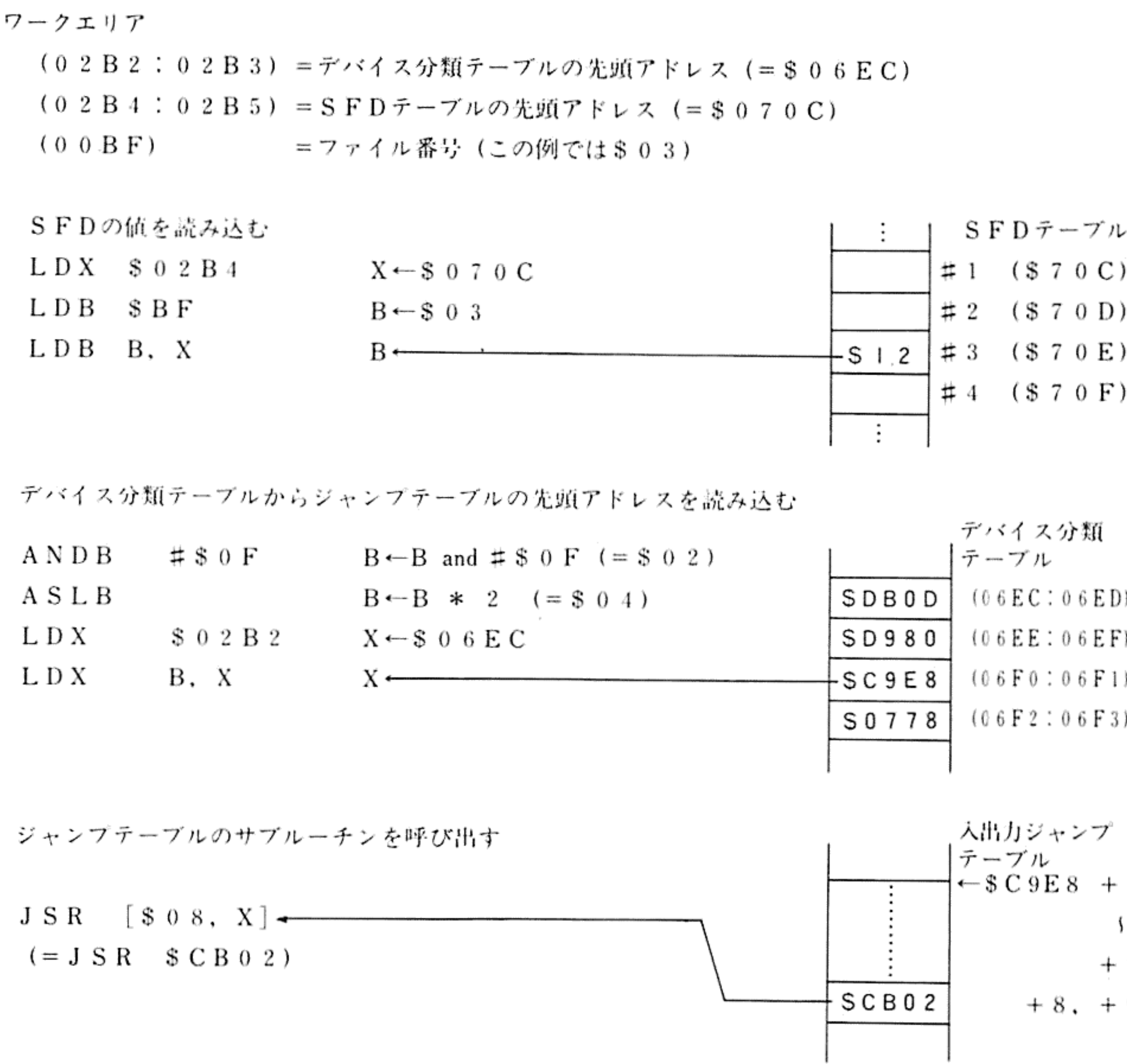
（1）ＳＦＤの値を読み込む

```
LDX  $02B4     X←$070C
LDB  $BF       B←$03
LDB  B,X       B←───────────── $12
```


（2）デバイス分類テーブルからジャンプテーブルの先頭アドレスを読み込む

```
ANDB  #$0F     B←B and #$0F (=$02)
ASLB           B←B * 2  (=$04)
LDX   $02B2    X←$06EC
LDX   B,X      X←───────────── $C9E8
```


（3）ジャンプテーブルのサブルーチンを呼び出す

```
JSR   [$08,X]←──────
(=JSR  $CB02)
```

4—2　入出力応用ルーチンの概要

4—2—1　セーブ（SAVE）ルーチンの概要

アドレス　＄CD3E

解　　説　セーブには4種類あり，それぞれのタイプに分けて説明します．

（1）　SAVEM　＄CE04

●＄CD0Eを呼び，ファイル名・デバイスの選択，ファイル番号＃17を設定，ファイルタイプ，アスキーフラグ，テープファイルモードのクリアを行います．

●ファイルタイプを＄02に指定した後，＄CEDFでオープン処理を行い，＄00・プログラムの長さ・開始番地・プログラムの内容・＄FF・＄0000・入口番地の順に出力します．

●＄CE4Bのクローズ実行ルーチンに入り，クローズ処理をします．

（2）　SAVE　＄CD3E→＄CDCF

●＄CD0Eを呼んだ後，＄CDCFにジャンプし，プロテクトチェック（＄CDC6）とオープン処理（＄CEDF）を行います．

●＄CDD6からプログラムの出力を行い，＄FF・（01E7）・（01E8）・テキストエリア〔先頭(33：34)～最終(35：36)〕の内容を出力した後，＄CE4Bのクローズ処理ルーチンへジャンプします．

（3）　SAVE　，A　＄CD3E→＄CD20

●＄CD0Eを呼んだ後，＄CD20にジャンプし，＄CD24からLISTとの共通ルーチンになるため＄9F0Bを呼び，LISTの範囲を検索します．

●＄CDC6を呼び，プロテクトチェックを行った後，アスキーフラグとテープファイルモードを反転させ，（BF）が0でなければ＄CEDFのオープン処理を呼び出します．

●＄9ED1のLIST出力ルーチンで，プログラムの出力を行います．

（4）　SAVE　，P　＄CD3E

●＄CD0Eを呼んだ後，＄CEDFを呼んでオープン処理をします．

●＄CD5Bからは，プロテクト変換をテキストエリアに対して行い，＄CDA6→＄CDD6のSAVE処理系を用い，最初の＄FFが＄FEに変わる（プロテクト指定）他は（2）と同様に出力します．

●＄CDAAから，プロテクトの逆変換を行います〔プロテクト出力の最中にブレーク・キーを押した場合，テキストはプロテクトがかかったまま使用不可の状態となるので（ディスクの場合は起こりません），このルーチンを呼んで下さい〕．

4—2—2　ロード（ＬＯＡＤ）ルーチンの概要

アドレス　＄ＣＦ２２，＄ＣＦ２９，＄ＣＦ３１

解　説　エントリの違いなどにより，次のワークエリアが変更されます．

- （０２ＥＥ）ＲＵＮ（＄ＣＦ２２）…＄０２，ＭＥＲＧＥ（＄ＣＦ２９）・ＬＯＡＤ・ＬＯＡＤＭ（＄ＣＦ３１）…＄００
- （０２ＥＦ）ＲＵＮ・ＬＯＡＤ・ＬＯＡＤＭ…＄００，ＭＥＲＧＥ…＄ＦＦ
- （０２Ｆ０）ＬＯＡＤＭ…＄００，それ以外…＄００以外の値
- （０２Ｆ１：０２Ｆ２）ＲＵＮ…＄０２００，ＬＯＡＤ・ＬＯＡＤＭ…＄００００

次に，共通ルーチンが始まる＄ＣＦ４Ｂからの流れを見ることにします．

- ＄ＣＣ３７でファイル名・デバイスの選択を行います．
- ＬＯＡＤＭでオフセットの指定があれば（０２Ｆ１：０２Ｆ２）に格納します．
- ＬＯＡＤ，ＬＯＡＤＭでR指定があれば（０２ＥＥ）に＄０３を代入します．
- （ここからＣＨＡＩＮとも共通ルーチンになりますが）デバイスがカセット・ディスク以外はエラーにします．（０２ＥＥ）が＄０３のものは，全てのファイルをクローズします（＄ＣＥ８１）．
- プログラムのファイル番号＃１７をオープンさせます（＄ＣＥＤＣ）．
- ＬＯＡＤＭなら＄Ｄ０３０にジャンプします．また，ＢＡＳＩＣソースのバイナリ形式のものは＄ＣＦＣ８にジャンプします．

以下，各場合に分けて説明します．

（１）　ＢＡＳＩＣソース，アスキー形式の場合

- アスキーフラグとテープファイルモードのチェックを行います．
- ＭＥＲＧＥ指定がないと，テキスト消去・クリア処理(＄８Ｆ３９)をします．
- テキスト作成ルーチン（＄８Ｅ８８）にジャンプします．

（２）　ＢＡＳＩＣソース，バイナリ形式の場合

- ＭＥＲＧＥ指定があれば，エラーにします．
- ＄８Ｆ３９を呼んだ後，３バイト入力し（Ｄ１），（０１Ｅ７：０１Ｅ８）に格納します．※（Ｄ１）へは，入力したデータに＋１した値が入ります．
- ＄８ＤＡＥのメモリフルテストを行いながら，データ終了フラグ（Ｃ０）が立つまで,（３３：３４）から格納していきます．この際，（３５：３６）が順次更新されます．

●プロテクトの場合は，プロテクト逆変換を行います（＄ＣＤＡＡ）．
これは，１バイト目が＄ＦＥのとき実行されます．
●プログラム最終部の，リンクポインタを修正します．
●（ここからは（1）と共通になりますが）クローズ処理（＄ＣＥ４Ｂ），変数消去（＄８Ｆ４Ｂ），リンクポインタ修正（＄Ｃ７３０）を行います．ＲＵＮ指定があれば実行解読ルーチン（＄Ｃ６３Ｄ），なければメインルーチンプロンプト表示部（＄８Ｅ７２；Ready 出力）へジャンプします．

（３）　ＬＯＡＤＭの場合

●ファイルタイプとアスキーフラグのチェックを行います．
●プログラムの長さと先頭アドレスを読み込み，オフセットを加えたアドレスから，長さ分だけのデータをメモリに格納した後，入口番地を（０２１７：０２１８）に設定し，クローズ処理（＄ＣＥ４Ｂ）を行います．Ｒ指定があればＥＸＥＣ（＄Ｃ７７７）に飛び，プログラムの実行を開始します．

4―2―3　オープン（ＯＰＥＮ）ルーチンの概要

アドレス　＄ＣＥＡ８～＄ＣＦ２１
（＄ＣＥＤＣ，＄ＣＥＤＦ，＄ＣＥＥ１～＄ＣＦ２１）

解　説　データファイルをオープンさせるこのルーチンは，次のような手順で実行されます．

（1）　テキストを読み込み，ファイルのモードのアスキーコードをスタックに設定します（＄９２Ｃ３，＄９９４Ｂを使用）．
（2）　テキストのファイル番号を読み込み（＄ＣＥ８Ｄ），そのファイル番号のＳＦＤがオープン状態にないことを調べます．
（3）　テキストからファイルディスクリプタを読み込み，ワークエリアにファイル名，オプション，デバイス番号を設定します（＄ＣＣ３７）．
（4）　ファイルタイプ（０２ＤＢ）をＢＡＳＩＣデータ（＝＄０１）に設定し，アスキーフラグ（０２ＤＣ）とカセット・テープファイルモード（０２Ｄ４）をアスキー（＝＄ＦＦ）に設定します．
（5）　オープン実行ルーチン（＄ＣＥＥ１）を呼び，各デバイスをオープンさせた後，エラーがなければファイル番号をクリアして終了します．

オープン実行ルーチンを使用する際の内容は，以下のようになります．

- **＄ＣＥＤＣ…インプットモード用のエントリ（ＬＯＡＤなどで使われる）**
- **＄ＣＥＤＦ…アウトプットモード用のエントリ（ＳＡＶＥなどで使われる）**
- **＄ＣＥＥ１…Ｂレジスタに，そのファイルのモードをアスキーコードで入れておくことが必要**

実行ルーチンの入力情報としてオープンルーチンが設定するものを与えます．

（００ＢＦ）＝ファイル番号（１～１７）
（０２Ｄ４），（０２ＤＣ）＝アスキーフラグ
（０２ＤＢ）＝ファイルタイプ
} **インプットモード時には復帰情報になります．**
（０２ＤＤ）＝ファイル名の長さ（０～８）
（０２ＤＥ～０２Ｅ５）＝ファイル名
（０２Ｅ６）＝デバイス番号
（０２Ｅ７～０２ＥＣ）＝ファイルのオプション

なお，このルーチンではファイルモード(０２ＤＡ)を設定し，その内容と（０２Ｅ６）のＯＲを取って，そのファイル番号のＳＦＤを設定します．

４―２―４　クローズ（ＣＬＯＳＥ）ルーチンの概要

アドレス　**＄ＣＥ７３，＄ＣＥ８１～＄ＣＥ８Ｃ**
（＄ＣＥ４Ｂ，＄ＣＥ４Ｄ～＄ＣＥ７２）

解　　説　特定のファイルをクローズする場合と，全てのファイルをクローズする場合との２種類があります．

（１）　テキストの「ＣＬＯＳＥ」の後ろにファイル番号が書かれていない場合には，＄ＣＥ８１にジャンプします．＄ＣＥ８１では，＃０～＃１７のファイルを全てクローズさせます（＄ＣＥ４Ｄ使用）．

（２）　ファイル番号が書かれている場合には，ファイル番号を（００ＢＦ）に設定し（＄ＣＥ９０），そのファイルをクローズさせます（＄ＣＥ４Ｂ使用）．ファイル番号が複数書かれている場合には，「，」をチェックしてから，再びファイル番号を設定（＄ＣＥ８Ｄ）し，同様にクローズさせます．

次に，クローズ実行ルーチンについて解説します．

- **＄ＣＥ４Ｂ…（００ＢＦ）の示すファイル番号のファイルをクローズします．**

●＄ＣＥ４Ｄ…Ｂレジスタの示すファイル番号のファイルをクローズします．

その処理は，以下のようになります．

（1）　そのファイル番号の示すＳＦＤを参照し，デバイスがディスクのときは，ディスクのクローズルーチンを呼びます．

（2）　ディスクでないときは，（０２ＤＡ）にファイルのモードを入れ（ＳＦＤから求める），ＳＦＤの指し示す，デバイスのクローズルーチンを呼びます．

（3）　そのＳＦＤの内容をクリアします．

4—3　基本的入出力ルーチン

基本的入出力ルーチンは，４－３－４節のＦＩＲＱおよびAbortルーチンを除いては，全てＳＦＤ（システムファイルディスクリプタ）とファイル番号（００ＢＦ）を設定しておくことが前提となっています．設定されていない場合，即ち，その両方のうちいずれかが，または両方の値が＄００になっているときは，コンソールへの入出力とみなし，その処理を行います．

4—3—1　ジャンプテーブルを使う基本的なルーチン

これらのサブルーチンは，図４・１・１２に示したような形でＳＦＤの指すデバイスのサブルーチンを呼び出してきます．

（1）　1バイト入力ルーチン

アドレス　＄Ｄ０７２～＄Ｄ０８Ｄ

機　　能　そのファイルのＳＦＤ（システムファイルディスクリプタ）の示すデバイスから，１バイト入力する．

レジスタ　Ａ，Ｂ，Ｘ　（Ｂ，Ｘ，Ｙ，Ｕは保存）

復帰情報　Ａレジスタ　＝入力データ
（００Ｃ０）＝ファイルの終わり，またはそれ以降に入力を行った場合は，＄ＦＦが設定される（それ以外は＄００）．

WORK　（００ＢＦ）　Ⓡ(＝Read)，　（００Ｃ０）　Ⓡ
（０２Ｂ２：０２Ｂ３）　Ⓡ　，（０２Ｂ４：０２Ｂ５）　Ⓡ

解　　説　入力データ終了フラグ（００Ｃ０）をクリアした後，ＳＦＤテーブルの先頭アドレス（０２Ｂ４：０２Ｂ５）とファイル番号（００ＢＦ）から，目的のＳＦＤを見つけ，ＳＦＤの示すデバイスがディスクのときは，図４・１・１１で示されるアドレスへジャンプし，その他のデバイスは，デバイス分類テーブル（図４・１・９）を参照し，そのジャンプテーブル（図４・１・１０）で示されるサブルーチンを呼び出します（図４・１・１２参照）．入力は，これらのサブルーチンが行います．

(2)　1バイト出力ルーチン

アドレス	$D08E～$D0C9
機　能	そのファイルのSFDの示すデバイスに，1バイト出力する．
レジスタ	A，B，X　（A，B，X，Y，Uは保存）
入力情報	Aレジスタ＝出力データ
WORK	(00BF) Ⓡ　　　，(02B2：02B3) Ⓡ (02B4：02B5) Ⓡ　，(05AA：05AB) Ⓡ (05AC) Ⓡ
SUB	$D617→プリンタ1バイト出力ルーチン

解　説　ファイル番号の指定がないとき〔(00BF)＝0〕で，かつ，プリンタONフラグが立っているとき〔(05AC)≠0〕は，Aレジスタのアスキーコードに対応する文字をプリンタへ出力することができます．また，そのファイルのSFDが，インプットモードで，かつ，コンソール以外のデバイスを指しているときは出力しません．その他の場合は，$D072と同じ方法で，SFDの示すデバイスへ出力します（図4・1・9～図4・1・12参照）．

(3)　ポジション・デバイスチェックルーチン

アドレス	$D0CA～$D0EC
機　能	カーソルやプリンタヘッドのポジションなど，改行するために必要な情報を与える．
レジスタ	A，B，X　（全て保存）
復帰情報	Zフラグ＝カセットおよびディスクのときはセットされる． ※ワークエリアについては，図4・3・1を参照．
WORK	(00BF) Ⓡ　　　，(00CD～00D0) Ⓦ (02B2：02B3) Ⓡ　，(02B4：02B5) Ⓡ (02D9) Ⓦ(＝Write)

解　説　$D072と同じ方法で，SFDの示すデバイスのサブルーチンを呼び，図4・3・1に示したようにワークエリアの設定を行います．

図4・3・1　$D0CAによって設定されるワークエリア

ワークエリア デバイス ［サブルーチンのエントリ］	(00CD)	(00CE)	(00CF)	(00D0)	(02D9)
KYBD,SCRN 〔$D992〕	$0E	(0311) フィールド改行桁数	(030B) カーソルのX座標	(00C3) WIDTH幅	$00
CAS0 〔$CB4D〕	$01	$00	$00	$00	$FF
LPT0 〔$D39C〕	$0E	(X+$14) フィールド改行桁数	(X+$12) 現在の桁位置	(X+$13) 改行桁数	$00
COMn 〔$D39C〕	$0E	(X+$14) フィールド改行桁数	(X+$12) バッファ内のデータ数	(X+$13) 改行桁数	$00
DISK 〔$7AA8〕	$0E	$00	(FBUF+6) バッファ内のデータ数	$00	$00

※( ) で囲まれているものは，ワークエリアまたはバッファ内のメモリであることを示しています．また，( ) 内のXは，そのデバイスのジャンプテーブルの先頭アドレスを示し，FBUFはファイルバッファの先頭アドレスを示しています．

各ワークエリアの意味

(00CD)　**＝フィールドの桁数**；コンマで区切ったときのフィールドの桁数，あるいは，コンマで移動する桁数を示します．

(00CE)　**＝フィールド改行桁数**；コンマで区切った場合，現在の桁位置がこの値より大きくなった場合は改行されます．

(00CF)　**＝現在のポジション**；現在のデータ内の位置を示します．

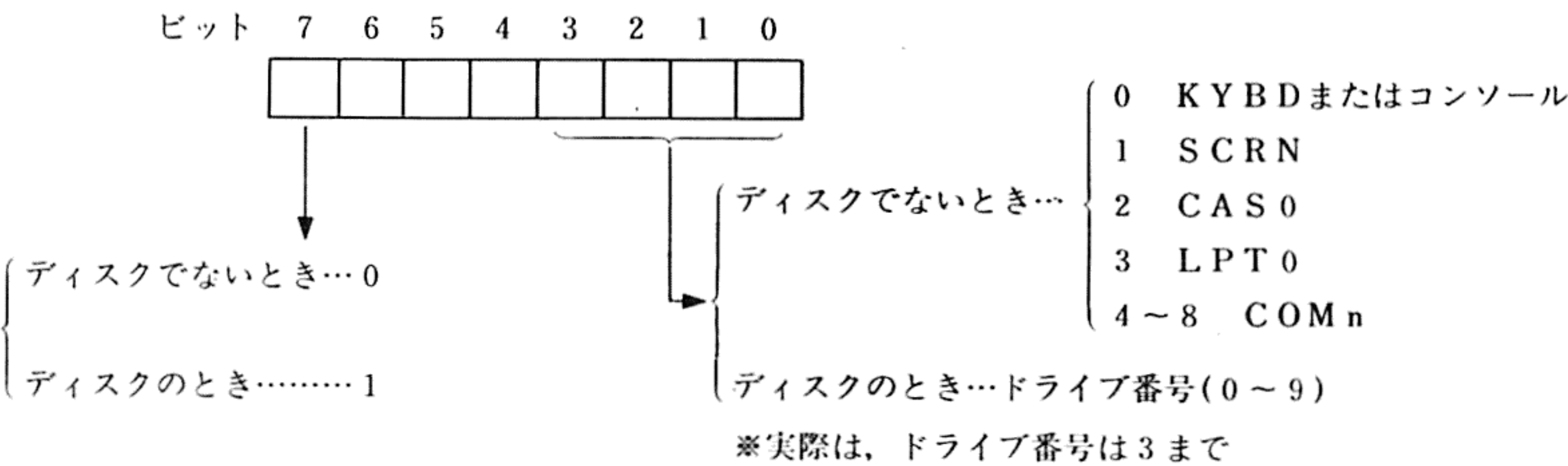
(00D0)　**＝改行桁数**；改行すべき桁数を示します．

(02D9)　**＝LF出力禁止フラグ**；改行の際$0A (LF) を出力しないデバイスを示します．

図4・3・2　デバイス番号（02E6）

ビット　7　6　5　4　3　2　1　0

ビット7：
- ディスクでないとき…0
- ディスクのとき………1

ビット3〜0：
- ディスクでないとき…
  - 0　KYBDまたはコンソール
  - 1　SCRN
  - 2　CAS0
  - 3　LPT0
  - 4〜8　COMn
- ディスクのとき…ドライブ番号(0〜9)
  ※実際は，ドライブ番号は3まで

4—3—2 ファイル関係ルーチン

(1) ファイル名，デバイス設定ルーチン

アドレス　$CC34，$CC37，$CC3C，$CCAC～$CD00

機　　能　テキストの文字列式からファイルディスクリプタ（FD）を読み込み，ファイル名とデバイス番号をワークエリアに格納する．

復帰情報　解説の下にあるワークエリアが設定される．

WORK　(01EB) Ⓡ　，(02B2：02B3) Ⓡ
(02DD) Ⓦ　，(02DE～02E5) Ⓦ
(02E6) Ⓦ　，(02E7～02ED) Ⓦ

S U B　$00D8→汎用読み込みルーチン
$0299→拡張用
$98F1→式の評価・型判断と文字列の実行アドレスを読み込む

終了条件　指定したデバイスがなかったとき“Device Unavailable”エラー（$CF0A），ファイルディスクリプタの記述の仕方が間違っているとき“Bad File Descriptor”エラー（$CCFC）を発生．

解　　説　$CC34は「FILES」，$CC3Cは「LOAD?」用のエントリで，FDを必ずしも必要としません．その他のFDを必要とするもの（ROMモードでの「LOAD」も原則的に含まれます）は$CC37をエントリとしています．これらは，式の評価をしてFDを表わす文字列の文字数と先頭アドレスを設定（$98F1）し，それを基にディスクは$CC6Cで，その他のデバイスは$CCC5～$CCFB間で，デバイスを選択します．$CC83からはオプションを，$CCA5からはファイル名を設定する部分です．また，$CCACからはファイル名転送ルーチンとしても呼ばれます．ユーザが$CC37を使用するときは引用符で始まるFDの先頭アドレスを（D9：DA)に代入して下さい．

- (02DD)　=ファイルの長さ；（$00～$08の間の値）
- (02DE～02E5)　=ファイル名；アスキーコードで入る．
- (02E6)　=デバイス番号；デバイスの指定のないときは，デフォルトのデバイス番号（01EB）が代入される．
- (02E7～02ED)　=ファイルのオプション；アスキーコードで入る．

(2)　ファイル番号設定ルーチン

アドレス	＄CE90～＄CEA6
機　　能	テキストから読み込み，ファイル番号（00BF）を設定する．
レジスタ	A，B
WORK	（00BF）　Ⓦ
SUB	＄00D2，＄00D8→汎用読み込みルーチン ＄99BC→式の評価から1バイト整数データを作ってBレジスタに入れるルーチン
終了条件	ファイル番号が1以上16以下でないと"Bad File Number"エラーを発生させるためにエラー処理ルーチン（＄8DD1）に飛ぶ．

(3)　ファイルチェックルーチン

アドレス	＄D0ED～＄D0F9
機　　能	SFDを調査し，デバイス情報をCCレジスタに設定する．
レジスタ	B，X　（全て保存）
復帰情報	Nフラグ＝デバイスがディスクのときはセットされる． Zフラグ＝そのファイルがオープンされていないときはセットされる．
WORK	（00BF）　Ⓡ　，（02B4：02B5）　Ⓡ

(4)　ファイルモードチェックルーチン

アドレス	＄D0FA，＄D0FD～＄D119
機　　能	SFDを調査し，そのファイルのモードのチェックを行う． アウトプット（＄D0FA）／インプット（＄D0FD）
レジスタ	A，B，X　（B，Xが保存）
復帰情報	Aレジスタ＝アウトプット…＄20／インプット…＄10
WORK	（00BF）　Ⓡ　，（02B4：02B5）　Ⓡ
終了条件	ファイルがオープンされていないときは，"File not Open" エラーを発生させる．指定したモードと異なるときは，"Bad File Mode" エラー（＄CEF4）を発生する． ※ファイル番号（00BF）が0のときは，チェックを行わない．

4—3—3　基本的入出力ルーチンを利用したデータ入出力ルーチン

(1)　ブロック出力ルーチン

アドレス　$D90F～$D92E

機　能　コンソールあるいはそれ以外のデバイスに，255バイトまたはデータ中に$00がくるまで，データを出力する．

レジスタ　A，B，X

入力情報　Xレジスタ＝データの先頭アドレス

WORK　(00BF)＝ファイル番号　Ⓡ

SUB　$D08E→1バイト出力ルーチン
$DA30→SUBOUT　PUT初期設定ルーチン
$DA36→SUBOUT　PUT後続設定ルーチン
$DA64→BIOS実行ルーチン

解　説　次の2とおりの動作をします．

(i)　ファイル番号が設定されている場合〔(00BF)≠0〕は，$D08Eを用いてSFDの示すデバイスに出力します．

(ii)　ファイル番号が設定されていない場合〔(00BF)＝0〕は，$DA30，$DA36，$DA64を用いて画面上に出力します．その際，画面上のフィールドは設定されません．

(2)　ブロック入力ルーチン

アドレス　$DAA6，$DAB2～$DADE

機　能　システム入力用バッファ(043D～053C)に255バイト，またはデータ中に$0Dがくるまで，データをデバイスから入力する．

レジスタ　A，X

復帰情報　Xレジスタ　＝$043Cが代入される．
(00C0)＝ファイル終了の場合は，$FFが設定される．

WORK　(00C0)＝データ終了フラグ　Ⓦ

SUB　$0260→拡張用

＄Ｄ０７２→１バイト入力ルーチン
＄Ｄ０ＥＤ→ファイルチェックルーチン

解　説　＄ＤＡＡ６エントリのとき，ＳＦＤが設定されていない場合には何も行いません（Ｚフラグがセットされます）．また，ＳＦＤが設定されている場合はスタックポインタを＋２して，このサブルーチンを呼び出したＪＳＲ・ＢＳＲ命令の書かれている番地を放棄するので，このルーチンが終わると，２段階上位のルーチンへリターンします．ユーザがこのルーチンを使用する場合は，これらの処理を行わない＄ＤＡＢ２をエントリとするのが適当です．なお，入力したデータの後尾には，データの終了を示すため＄００が付加されます(従って，データが実際に入るのは＄０５３Ｂまでとなります．また，このルーチンを使ってキーボードから入力する際は，画面に入力文字が出力されません）．

（３）　一行出力ルーチン

アドレス　＄９ＢＤＢ，＄９ＢＦ３，＄９ＢＦ６～＄９Ｃ２Ａ
機　能　コンソール，あるいはその他のデバイスに，文字列・データを出力する．
レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報　解説参照
復帰情報　Ｂレジスタ＝クリアされる（Ｚフラグもセットされる）．
Ｘレジスタ＝文字列（データ群）の最終アドレス＋１の値
ＷＯＲＫ　（００ＢＦ）　Ⓡ　，（０５Ａ０～０５Ａ５）　Ⓦ
（０５ＡＣ）　Ⓡ
ＳＵＢ　＄９７９Ｂ→文字列定数などの読み出し評価ルーチン
＄９８Ｆ７→文字列の実行アドレス読み込みＳＡＣを１つつぶす
＄９Ｂ５０→改行ルーチン
＄Ｄ０８Ｅ→１バイト出力ルーチン
＄Ｄ９２Ｆ→フィールド設定ルーチン
＄００ＤＥ→ＢＩＯＳジャンプルーチン

解　説　エントリによって，処理内容および入力情報が多少異なってきます．
● ＄９ＢＤＢ　Ｘレジスタに文字列の先頭アドレス－１の値，文字列の終わりには＄００，もしくは＄２２を入れて下さい．

●＄９ＢＦ３　ＰＲＩＮＴ文やＩＮＰＵＴ文プリント用のエントリ（このエントリから入るときは，ＳＤ（ストリングディスクリプタ）などを正しく設定しておく必要があります．

●＄９ＢＦ６　Ｂレジスタに文字列の長さ，Ｘレジスタに文字列の先頭アドレスを設定して下さい．

＄９ＢＤＢエントリの場合は，文字列の先頭に＄０Ｄ，＄０Ａの並びがあるときには，１行改行（＄９Ｂ５０）します．これ以降の動作は，ＳＡＣ（ストリングアキュムレータ；３章参照）を設定（＄９７９Ｂ）し，そのＳＡＣを基に，文字列の長さと実行アドレスを設定（＄９８Ｆ７）します．ここまでが＄９ＢＦ６までの動作で，＄９ＢＦ６以降は，ファイルとしてでないコンソールへの出力の場合〔（００ＢＦ）＝０〕は，画面上のフィールド（システム仕様２．２．４（５））を設定（＄Ｄ９２Ｆ）します．このとき，プリンタＯＮフラグが立っていない場合〔（０５ＡＣ）＝０〕は，ＢＩＯＳのＯＵＴＰＵＴ〔システム仕様２．１．４（１６）〕を用いて文字列のコンソールへの出力を行います．その他の場合は，１バイト出力ルーチン（＄Ｄ０８Ｅ）を用いて，ＳＦＤが指定しているコンソールやその他のデバイスに文字列を出力します（この動作は，＄９Ｃ１６から行われますから，ここをエントリとしても構いません）．

（４）　改行ルーチン

アドレス	＄９Ｂ４７，＄９Ｂ５０〜＄９Ｂ６７
機　能	デバイスに＄０Ｄ（ＣＲ）と＄０Ａ（ＬＦ）を出力します．
レジスタ	Ａ，Ｂ，Ｙ
ＷＯＲＫ	（００ＣＦ）Ⓡ　，（０２Ｄ９）Ⓡ
ＳＵＢ	＄Ｄ０８Ｅ→１バイト出力ルーチン ＄Ｄ０ＣＡ→ポジション・デバイスチェックルーチン ＄Ｄ６４Ａ→プリンタ改行ルーチン

解　説　＄９Ｂ４７エントリの際には，ディスク，カセット以外のデバイスは，Ｘ座標が０のときは改行しません．改行する際に，デバイスがプリンタのときは＄Ｄ６４Ａを用いて改行します．また，デバイスがカセットのときは，＄０Ａ（ＬＦ）が出力されません．なおワークエリアは，図４・３・１を御参照下さい．

(5) 一行入力ルーチン ——スクリーン エディタ——

アドレス	$D7F7, $D807～$D8AB

機　能	キーボード，あるいは周辺装置から1行を入力して，システム入力

用バッファ（043D～053C）に格納する．

レジスタ	A，B，X，U

復帰情報	キーボードから入力したとき，CNTL－X，CNTL－C，あるいはブレーク・キーで終了した場合は，Cフラグがセットされる．リターン・キーで終了した場合は，Xレジスタに$043C，Uレジスタにバッファ内の文字列の最終アドレスが，各々代入される．

WORK	(D9：DA) Ⓦ　，(0312：0313) Ⓡ／Ⓦ (05A9) Ⓡ／Ⓦ　，(05B7) Ⓡ／Ⓦ

SUB	$00D2→汎用読み込みルーチン $00DE→BIOSジャンプルーチン $9B47，$9B50→改行ルーチン $D93D→フィールド設定およびEL(イレーズライン)ルーチン $D9F4→汎用入出力RCB設定ルーチン $DAA6→ブロック入力ルーチン $DAEF，$DAF6→カーソル表示，消去ルーチン

解　説	このルーチンでは，両方のエントリの最初で$DAA6を呼んでい

るため，ファイル番号（00BF）とSFDが設定されている場合は，そこで周辺装置からの入力を行い，このルーチンで入力を行いません．

コンソールからの入力の場合は，BIOSのINPUT，あるいはINPUTC〔システム仕様2.1.4(14)，(15)参照〕を用います．ここで注意することは，FM－7は，画面をフィールドごとに分割して扱っているということです．従って，カーソルを移動する等で，複数のフィールドを変更したときは，変更されたフィールド全てがコンソールからの入力となります．ところがINPUTを使用して持ってこられるものは，最初のフィールドのみで，残りのフィールドは，INPUTCを使って1回ごとに持ってくる必要があります．

しかし，他のフィールドを変更したときに，インタプリタがそれらのフィールドを必要とするのは，次の2つの場合しかありません．

● コマンドレベル（コマンド待ちの状態）で，画面上に出ている，以前入力したコマンドを修正して使用した場合

●コマンドレベルで，画面上に出ている複数のプログラムの行を変更した場合

このような場合は，ＩＮＰＵＴで提出されたフィールド以外の，変更されたフィールドをＩＮＰＵＴＣで持ってくる必要があります．従って，この＄Ｄ７Ｆ７では，ＩＮＰＵＴで提出されたフィールドに何の入力もないなら，ＩＮＰＵＴＣを用いて，変更された次のフィールドをバッファに持ってきます．また，ＩＮＰＵＴで提出されたフィールドの先頭が数字で始まるのなら，すなわちプログラムの行の編集の際は，**インプット継続フラグ**（０５Ａ９）を立てたままにしておいて（＝＄ＦＦ），次の機会に引き続いて，このルーチンを呼び出すときにはＩＮＰＵＴＣで残りのフィールドを転送するようにします．

このように，＄Ｄ７Ｆ７のエントリでこのルーチンを呼ぶときは，インプット継続フラグにより，ＢＩＯＳのＩＮＰＵＴ，あるいはＩＮＰＵＴＣを用いますが，＄Ｄ８０７のエントリで呼ぶときは，ＩＮＰＵＴのみを用いており，またインプット継続フラグを立てないので，上記の動作はしません．

※ＩＮＰＵＴ，ＩＮＰＵＴＣでは，変更されたフィールドを画面最上段のフィールドより順番に読み込んできます．

バッファに転送されたフィールド内の文字列の中の一文字のアスキーコードが＄００～＄１Ｆ，あるいは＄７Ｆ，＄ＦＥ，＄ＦＦであった場合は，スペース（＝＄２０）で置き換えられます．また，文字列の最後にスペースがある場合は，そのスペースは無視されます．

一連の入力が終わったときに，Ｘ座標が０でなければ改行をします（＄９Ｂ４７）．また，ブレーク・キーやＣＮＴＬ－Ｘ，ＣＮＴＬ－Ｃなどのキーによって入力が終了した場合は，インプット継続フラグ（０５Ａ９）とブレーク・キー押下フラグ（０３１２：０３１３）をクリアした後，改行をして（＄９Ｂ５０）終わるだけで，バッファに転送されません．

※文中で使用した「バッファ」というのは，**システム入力用バッファ**（０４３Ｄ～０５３Ｃ)〔第２章で「行入力バッファ」と呼んだもの〕のことを指しています．

4―3―4　ＦＩＲＱと Abort ルーチン

（1）　ＦＩＲＱルーチン

アドレス　＄Ｃ９５３～＄Ｃ９Ｄ７

レジスタ　Ａ，Ｂ，Ｘ〔全てのレジスタ（ＣＣ，Ｓ，ＤＰも含む）が保存〕

解　　説　ここでは，ブレーク・キーとの関連だけを述べることにします．

- ブレーク・キーの押下のチェックを行い，押されていれば，サブシステムにＩＲＱリクエストを出して命令の実行を解除させた後，ブレーク・キー押下フラグをセット〔(０３１３)←＃＄４０〕します．この後，ＩＲＱリクエストを解除させます．また，押されている間はループを繰り返しています．
- ブレーク・キーの割り込み以外のサブシステムの**アテンション**なら，その割り込みの処理をします．

（2）　Abort ルーチン

アドレス　＄Ｄ６７８～＄Ｄ６Ｄ７

解　　説　入出力の Abort テストを行います．

- 拡張フック（０２８Ｄ）を通した後，ブレーク・キー押下フラグ（０３１３）をチェックし，＄００（押されていない）なら，そのままリターンします．
- （０５ＡＣ），（０３１２：０３１３），（００ＢＦ），（００１５），ＤＰレジスタをクリアします．
- ＄８Ｆ８２を呼び，ＳＰの初期設定，ＣＯＮＴアドレスのクリア，ＳＡＣの初期設定を行った後，＄ＥＣ０５で音関係の初期化，＄ＤＢ４４でＢＥＥＰ　ＯＦＦ，＄Ｄ７７２でＭＯＴＯＲ　ＯＦＦを行います．
- （０２Ｆ３）のＬＯＡＤフラグが立っていれば，＄８Ｆ３９でテキスト消去，変数クリアなどの処理を行います．
- また，ＴＥＲＭモードのときは，割り込みの初期化やクローズ処理を行います．
- ＄８Ｅ６３のメインルーチンプロンプト出力部にジャンプし，「Abort」出力を行いメインルーチンに戻ります．

4—3—5　サンプル

ディスクやカセットのファイルを画面に出力します．プログラムファイルの場合はフィールドをセットしてあるので，画面に出たリストを修正するだけで，プログラムとして格納されます．また，カセットにバイナリ形式でセーブされたファイルはギャップが短いためブロックを読み飛ばすことがあるので，１ブロックを読んで表示している間にモータを少し戻しておくか，あるいは，あらかじめ長いギャップでセーブする〔(０１Ｅ９)を＄８０ぐらいにしておく〕かを行っておく必要があります．

```
PAGE  001 (821201,000255) F_LIST

10000                              NAM    F_LIST
10010                              OPT    MEM,NOGEN
10020  5000                        ORG    $5000
10030             5000      F_LIST EQU    *
10040  5000 0F    BF               CLR    $BF
10050  5002 30    8D 00CA          LEAX   MSG-1,PCR
10060  5006 BD    9BDB             JSR    $9BDB    *Print_out MESSAGE
10070  5009 BD    D807             JSR    $D807    *Line_input from KYBD
10080  500C 86    22               LDA    #$22     *Set Double Quotation Mark
10090  500E A7    84               STA    ,X       to Top of File Descriptor
10100  5010 DE    D9               LDU    $D9
10110  5012 34    40               PSHS   U        *Push (D9:DA)
10120  5014 9F    D9               STX    $D9
10130  5016 BD    CC37             JSR    $CC37    *Set File Name & Device#
10140  5019 0C    BF               INC    $BF      *File# = 1
10150  501B BD    CEDC             JSR    $CEDC    *Open File (Input Mode)
10160  501E B6    02DC             LDA    $02DC
10170  5021 27    1A   503D        BEQ    BINARY   *If Binary File then Binary
10180             5023      REPEAT EQU    *
10190  5023 BD    DAB2             JSR    $DAB2    *Block Input from file
10200  5026 96    C0               LDA    $C0
10210  5028 26    7B   50A5        BNE    ESC      *if End_of_File then Escape
10220  502A 0F    BF               CLR    $BF      *File# = 0 (Normal)
10230  502C 30    01               LEAX   1,X
10240  502E BD    D92F             JSR    $D92F    *Set Field
10250  5031 BD    D90F             JSR    $D90F    *Block Output to SCRN(Console)
10260  5034 BD    9B50             JSR    $9B50    *Output CR & LF
10270  5037 0C    BF               INC    $BF      *File# = 1
10280  5039 8D    78   50B3        BSR    BRKCHK
10290  503B 20    E6   5023        BRA    REPEAT
10300             503D      BINARY EQU    *
10310  503D 73    02D6             COM    $02D6    *Disposition of CAS0
10320  5040 7D    02DB             TST    $02DB
10330  5043 26    77   50BC        BNE    BFMERR   *if Machine_Prog then ERROR
10340  5045 8D    7F   50C6        BSR    INPUT
10350  5047 4C                     INCA            *Protected_Program ?
10360  5048 26    77   50C1        BNE    PPERR    *if Protected_Program then ERR
10370  504A 8D    7A   50C6        BSR    INPUT    *skip UNLIST_# (upper byte)
10380  504C 8D    78   50C6        BSR    INPUT    *skip UNLIST_# (lower byte)
10390             504E      LIST   EQU    *
10400  504E 8D    76   50C6        BSR    INPUT    *read Link_Pointer (upper byte
10410  5050 34    02               PSHS   A
10420  5052 8D    72   50C6        BSR    INPUT    *read Link_Pointer (lower byte
10430  5054 AA    E0               ORA    ,S+      *if Link_Pointer = $0000
10440  5056 27    4D   50A5        BEQ    ESC      then Escape (CLOSE & RETURN)
10450  5058 8D    6C   50C6        BSR    INPUT    *read Line_Number
10460  505A 1F    89               TFR    A,B
10470  505C 8D    68   50C6        BSR    INPUT
10480  505E 1E    89               EXG    A,B
10490  5060 0F    BF               CLR    $BF      *File# = 0 (Normal)
10500  5062 BD    B615             JSR    $B615    *Output Line_Number
10510  5065 BD    9C22             JSR    $9C22    *Output Space
```

```
10520  5068 0C    BF                   INC    $BF       *File# = 1
10530  506A 8E    033C                 LDX    #$033C
10540             506D         LIST1   EQU    *
10550  506D 8D    57    50C6           BSR    INPUT
10560  506F A7    80                   STA    ,X+
10570  5071 27    15    5088           BEQ    OUTPUT   *if End_Of_Line then OUTPUT
```

```
PAGE   002 (821201,000255) F_LIST

10580  5073 81    FE                   CMPA   #$FE      *if not CONST then Continue
10590  5075 26    F6    506D           BNE    LIST1
10600  5077 8D    4D    50C6           BSR    INPUT     *Disposition of CONST
10610  5079 A7    80                   STA    ,X+
10620  507B 84    0F                   ANDA   #$0F
10630  507D 1F    89                   TFR    A,B       *Breg = bytes of CONST
10640             507F         LIST2   EQU    *
10650  507F 8D    45    50C6           BSR    INPUT
10660  5081 A7    80                   STA    ,X+
10670  5083 5A                         DECB
10680  5084 26    F9    507F           BNE    LIST2
10690  5086 20    E5    506D           BRA    LIST1
10700                          *
10710             5088         OUTPUT  EQU    *
10720  5088 8E    033C                 LDX    #$033C    *Xreg = Text_Buffer begin
10730  508B 108E  043D                 LDY    #$043D    *Yreg = I/O_Buffer begin
10740  508F 0F    5F                   CLR    $5F
10750  5091 0F    BF                   CLR    $BF       *File# = 0 (Normal)
10760  5093 BD    C177                 JSR    $C177     *One Line Dis-transration
10770  5096 8E    043D                 LDX    #$043D
10780  5099 BD    D90F                 JSR    $D90F     *Output One Line
10790  509C BD    9B50                 JSR    $9B50     *Output CR & LF
10800  509F 0C    BF                   INC    $BF       *File# = 1
10810  50A1 8D    10    50B3           BSR    BRKCHK
10820  50A3 20    A9    504E           BRA    LIST
10830                          *
10840             50A5         ESC     EQU    *
10850  50A5 35    40                   PULS   U         *Pull (D9:DA)
10860  50A7 DF    D9                   STU    $D9
10870  50A9 6F    C0                   CLR    ,U+       *Clear Text_Buffer
10880  50AB 6F    C0                   CLR    ,U+
10890  50AD 6F    C0                   CLR    ,U+
10900             50AF         CLOSE   EQU    *
10910  50AF BD    CE4B                 JSR    $CE4B     *CLOSE #1
10920  50B2 39                 RET     RTS              *Return from Subroutine
10930                          *
10940             50B3         BRKCHK  EQU    *
10950  50B3 FC    0312                 LDD    $0312     *Break_key pushed ?
10960  50B6 27    FA    50B2           BEQ    RET       *if not pushed then RTS
10970  50B8 32    62                   LEAS   $02,S
10980  50BA 20    E9    50A5           BRA    ESC       *if pushed then Escape
10990                          *
11000             50BC         BFMERR  EQU    *
11010  50BC 8D    F1    50AF           BSR    CLOSE
11020  50BE 7E    CEF4                 JMP    $CEF4     *Bad File Mode ERROR
11030                          *
11040             50C1         PPERR   EQU    *
11050  50C1 8D    EC    50AF           BSR    CLOSE
11060  50C3 7E    CDCA                 JMP    $CDCA     *Protected Program ERROR
11070                          *
11080             50C6         INPUT   EQU    *
11090  50C6 BD    D072                 JSR    $D072     *Input 1 byte
11100  50C9 0D    C0                   TST    $C0
11110  50CB 27    E5    50B2           BEQ    RET       *if not End_of_File then RTS
11120  50CD 32    62                   LEAS   $02,S
11130  50CF 20    D4    50A5           BRA    ESC       *if End_of_File then Escape
11140                          *
11150             50D1         MSG     EQU    *
```

```
PAGE  003 (821201,000255) F_LIST

11160  50D1       46                  FCC     !File Name :!
11170  50DC       00                  FCB     $00
11180                                 END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=50DC
PROGRAM ENTRY ADDR=****

exec&H5000
File Name :1:message

10 PRINT"In case of casset binary file , this program doesn't run correctly ,"
20 PRINT" because the gap between block and block is short ,"
30 PRINT" and the motor stop over the gap ."
40 PRINT"So, save binary file with long gap in advance ."
50 PRINT" (01E9)  #$0A ---> #$80"
60 PRINT"Or, push REV switch for a moment every gap ."


Ready
```

※ ディスクエディタ（4－5－6サンプル）の実行例

```
RUN
DRIVE :1
TRACK :1
SECTOR:1

DISK EDITOR FOR FM-7
          DRIVE : 1   TRACK : 1   SECTOR : 1
     +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00 : 00 FF FF FF FF C1 C0 C1 04 05 C6 08 C7 09 0A C0   :.....ﾁﾀﾁ..ﾆ.ﾇ..ﾀ
10 : 0C 0D 0E C0 C1 C0 C0 C1 C3 C5 C1 C1 C2 C3 1F 1B   :...ﾀﾁﾀﾀﾁﾃﾅﾁﾁﾂﾃ..
20 : C7 21 C1 C2 20 C1 22 C1 24 25 C2 27 28 C1 C1 C1   :ﾇ!ﾁﾂ ﾁ"ﾁ$%ﾂ'(ﾁﾁﾁ
30 : C0 C1 C0 C1 C0 C2 C2 C0 C0 C0 C0 C0 C5 C0 C1 C0   :ﾀﾁﾀﾁﾀﾂﾂﾀﾀﾀﾀﾀﾅﾀﾁﾀ
40 : C1 C0 C3 C3 C2 C0 C3 C4 C0 C0 C0 47 C5 49 4A 4B   :ﾁﾀﾃﾃﾂﾀﾃﾄﾀﾀﾀGﾅIJK
50 : C3 4D C5 FF FF FF FF FF FF FF FF FF FF FF FF FF   :ﾃMﾅ.............
60 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
70 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
80 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
90 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
A0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
B0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
C0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
D0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   :................
E0 : FF FF FF FF FF FF FF FF FF 66 66 66 66 66 66 66   :.........fffffff
F0 : 66 66 66 66 66 66 66 FF FF FF FF FF FF FF FF FF   :fffffff.........
KEY 'Q' - ESCAPE THIS MODE  'L' - HARDCOPY
```

4―4　カセット入出力ルーチン

4―4―1　カセット入出力ルーチンの概要とワークエリア

カセット入出力ルーチンのうち，上位ルーチンはカセット用バッファ（０５ＥＤ～０６ＥＢ）との入出力，下位ルーチンはＢＩＯＳを使って直接カセットとの入出力を行っています．上位ルーチンは，＄Ｃ９Ｅ８～＄Ｃ９Ｆ８のジャンプテーブルを先頭に，**＄Ｃ９ＦＡ～＄ＣＣ３３**に配置されており，下位ルーチンは**＄Ｄ６Ｄ８～＄Ｄ７Ｆ５**に配置されています．以下に各サブルーチンについて述べていきますが，ワークエリア名は下表に示すために省略します．なお，“バッファ”というのは４－４節ではカセット用バッファ（０５ＥＤ～０６ＥＢ）を指します．

表４・４・１　カセットの使用するワークエリア

ワークエリア	機能・解説
(０１Ｅ９)，(０１ＥＡ)	**ギャップの出力定数**；データの出力を行う前のモータの状態が，ＯＮなら（０１Ｅ９）＝＄０Ａ（１０）バイト，ＯＦＦなら（０１ＥＡ）＝＄ＦＦ（２５５）バイトのギャップ（＝＄ＦＦ）を出力します．
(０２Ｂ０：０２Ｂ１)	**カセット用バッファポインタ**；カセット用バッファの先頭アドレス（＄０５ＥＤ）の値を持っており，入出力にあたってこの値が参照されます．
(０２ＣＣ：０２ＣＤ)	**カセット入出力先頭アドレス**；カセットの１ブロックとの入出力を行うときの，データの先頭アドレスです．
(０２ＣＥ)	**ブロックのタイプ**；カセットの１ブロックのタイプで，カセットとの入出力の際に，ブロックの先頭に出力，あるいは入力されます（図４・４・１）．
(０２ＣＦ)	**ブロックの長さ**；この値が，入出力のブロックのバイト数を決定します（図４・４・１）．
(０２Ｄ０)	**カセット使用フラグ**；あるファイルがカセットを使用していることを示します．入力モード…＄０１　出力モード…＄０２
(０２Ｄ１)	**バッファ中のデータ数**；上位ルーチンがバッファをアクセスする際のバッファ中の有効データがどの位あるか示します．
(０２Ｄ２：０２Ｄ３)	**バッファ中のポインタ**；上位ルーチンがバッファをアクセスする際の，アクセスするデータのアドレスを示します．
(０２Ｄ４)	**テープファイルモード**；そのテープファイルのモードを示します．内容は，ファイルのアスキーフラグ（０２ＤＣ）と同様です．また，この内容が（０２Ｄ６）に代入されるため，アスキーフラグが立っていれば，１ブロック入出力の後ではモータがＯＦＦします．
(０２Ｄ５)	**エラーフラグ**；＄００…エラーなし　＄０１…チェックサム違いのエラー　＄０２…読み取り失敗のエラー
(０２Ｄ６)	**モータスイッチフラグ**；１ブロックの入出力が終わった後のモータの状態を決めます．　＄００…ＯＮ　＄ＦＦ…ＯＦＦ
(０２Ｄ７)	**モータ状態フラグ**；モータの状態を示します．　＄ＦＦ…ＯＮ　＄００…ＯＦＦ
(０２Ｄ８)	**ファイル名出力フラグ**；ファイルのヘッダブロックを読み込む際に，「Searching」や「Found」などのメッセージを画面に出力するかどうかを決めます．　＄００…出力する

カセット上のファイルのフォーマットは，文法書２．１０．１に書かれている通りですが，各々のブロックのフォーマットを下図に示しました．

図４・４・１　１ブロックの内容

ギャップ	$01	$3C			データ		$FF(4バイト)	

(＝$FF)
モータＯＮの状態では，
　１０バイト(０１Ｆ９)
ＯＦＦの状態では，
　２５５バイト(０１ＥＡ)

識別子

ブロックタイプ(０２ＣＥ)
　ヘッダブロック　$００
　データブロック　$０１
　エンドブロック　$ＦＦ

ブロックの長さ(０２ＣＦ)

(０２ＣＣ：０２ＣＤ)の示すアドレスから(０２ＣＦ)のバイト数のデータが入出される

チェックサム
色のついた部分の各々のバイトのΣ(総和)

図４・４・２　ヘッダブロックの内容

データ部　〔長さは$１４(＝２０[10])バイト〕

	$01	$3C	$00	$14	←8バイト→				←9バイト→		$FF(4バイト)	

(０２ＣＥ)
ブロックタイプ

(０２ＣＦ)
ブロックの長さ
(データ部の長さ)

(０２Ｄ４～０２Ｅ５)
ファイル名

(０２ＤＢ)
ファイルタイプ
　ＢＡＳＩＣソース　$００
　ＢＡＳＩＣデータ　$０１
　マシン語ファイル　$０２

(０２ＤＣ)
アスキーフラグ
　アスキー　$ＦＦ
　バイナリ　$００

(０２Ｄ４)
テープファイルモード
　プログラムファイル　バイナリ$００
　　　　　　　　　　　アスキー$ＦＦ
　データファイル　$ＦＦ

ブランク
$００

チェックサム
Σデータバイト

図４・４・３　エンドブロックの内容

	$01	$3C	$FF	$00	$FF	$FF(4バイト)	

(０２ＣＥ)
ブロックタイプ

(０２ＣＦ)
ブロックの長さ

チェックサム
ブロックの内容のΣ（総和），$ＦＦ＋$００＝$ＦＦとなっている．

図４・４・４　データブロックの内容

	$01	$3C	$01		データ		$FF(4バイト)	

(０２ＣＥ)
ブロックタイプ

(０２ＣＦ)
ブロックの長さ

(０２ＣＦ)の値が入出力するバイト数になる

チェックサム
ブロックの内容のΣ（総和）

4―4―2　カセット　オープン（ＯＰＥＮ）ルーチン

アドレス　＄ＣＡ０６～＄ＣＡ８Ｃ

機　　能　カセットに割り当てられたファイルをオープン（ＯＰＥＮ）する．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　Ａレジスタ　＝ファイルのモード（入力…＄１０，出力…＄２０）

（０２ＤＤ）＝ファイル名の長さ
（０２ＤＥ～０２Ｅ５）＝ファイル名
｝入力モードの場合は必ずしも設定しなくてもよい

（０２Ｄ４）＝テープファイルモード
（０２ＤＢ）＝ファイルタイプ
（０２ＤＣ）＝アスキーフラグ
｝入力モードの場合は復帰情報になる

復帰情報　（０２Ｄ０）＝カセット使用フラグ(入力…＄０１，出力…＄０２)

（０２Ｄ６）＝モータスイッチフラグ

ＷＯＲＫ　（０２Ｂ０：０２Ｂ１）Ⓡ，（０２ＣＣ：０２ＣＤ）Ⓦ
（０２ＣＥ）Ⓦ，（０２ＣＦ）Ⓦ
（０２Ｄ０）Ⓡ／Ⓦ，（０２Ｄ１）Ⓦ
（０２Ｄ２：０２Ｄ３）Ⓦ，（０２Ｄ４）Ⓡ／Ⓦ
（０２Ｄ６）Ⓦ，（０５ＥＤ～０６ＥＢ）Ⓡ／Ⓦ
（０２ＤＢ）Ⓡ／Ⓦ，（０２ＤＣ）Ⓡ／Ⓦ
（０２ＤＤ）Ⓡ，（０２ＤＥ～０２Ｅ５）Ⓡ

ＳＵＢ　＄ＣＡ７０→カセット用のポインタ初期化ルーチン
＄ＣＡ８Ｄ→カセット１ブロック入力ルーチン
＄ＣＢ５７→ヘッダブロックの入力とファイル名表示ルーチン
＄ＣＣＡＣ→ファイル名転送ルーチン
＄Ｄ７７２→モータＯＦＦルーチン
＄Ｄ７９４→カセット１ブロック出力ルーチン
＄ＥＣ０５→音関係初期化ルーチン（バッファのクリアを実行）

終了条件　カセットが使用中なら〔(０２Ｄ１）≒０〕，“Device In Use” エラー（＄ＣＡＡＥ）を発生させる．
出力モードでファイル名の長さがない〔（０２ＤＤ）＝０〕場合は “Bad File Descriptor” エラー（＄ＣＣＦＣ）を発生させる．
入力モードでヘッダの読み取りに失敗すると “Device I／O Error”（＄ＣＡＢ１）を発生させる．

解　説　動作内容は，出力モードと入力モードで異なる部分があります．

（１）　**出力モード，入力モード共通部分　＄ＣＡ０６～＄ＣＡ１７**

- カセット使用フラグ（０２Ｄ０）が立っていれば，エラーにします．
- ＰＬＡＹ文と同じバッファ領域（０５ＥＤ～０６ＥＢ）を使用しているため，バッファ領域と音関係の初期化（＄ＥＣ０５）を行います．

（２）　**出力モード　＄ＣＡ１８～＄ＣＡ７２**

- ファイル名（０２ＤＥ～０２Ｅ５），ファイルタイプ（０２ＤＢ），アスキーフラグ（０２ＤＣ），テープファイルモード（０２Ｄ４）をバッファ（０５ＥＤ～０６ＥＢ）に設定した後，カセットを動かし〔(ＦＤ００)←＄Ｃ２)〕，１０秒待った後にカセットに出力し（＄Ｄ７９４；図４・１・２参照），モータをＯＦＦし，カセット使用フラグ（０２Ｄ０）を立て，モータスイッチフラグ（０２Ｄ６）を設定します．
- バッファのポインタの初期化（＄ＣＡ７０）を行います．〔(０２Ｄ２：０２Ｄ３)←＄０５ＥＤ，（０２Ｄ１）←＄００〕．

（３）　**入力モード　＄ＣＡ７３～＄ＣＡ８Ｃ**

- ヘッダブロックをバッファに読み込み，ファイルの指定のある場合は目的のファイル名であるかを確認し（＄ＣＢ５７），バッファのポインタの初期化を行います（＄ＣＡ７０）．
- ファイルタイプ（０２ＤＢ），アスキーフラグ（０２ＤＣ），テープファイルモード（０２Ｄ４），モータスイッチフラグ（０２Ｄ６）を設定し，次の１ブロックをバッファに入力し（＄ＣＡ８Ｄ），カセット使用フラグ（０２Ｄ０）を立てます．

４―４―３　カセット　クローズ（ＣＬＯＳＥ）ルーチン

アドレス	＄ＣＡＢ６～＄ＣＡＥ３
機　能	カセットに割り当てられたファイルをクローズ（ＣＬＯＳＥ）する．
レジスタ	Ａ，Ｂ，Ｘ
入力情報	Ａレジスタ＝ファイルモード（入力…＄１０，出力…＄２０）
ＷＯＲＫ	（０２Ｄ０）Ⓦ　，（０２Ｄ１）Ⓡ
ＳＵＢ	＄ＣＢ３６，＄ＣＢ３８→１ブロック出力ルーチン ＄Ｄ７６Ｆ，＄Ｄ７７２→モータＯＮ，ＯＦＦルーチン

解　説　出力モードでは，事前に次の動作を行っています．

●バッファにまだ出力されていないデータが残っていれば（（02D1）≒0），データブロックとしてそれを出力（$CB36）します．

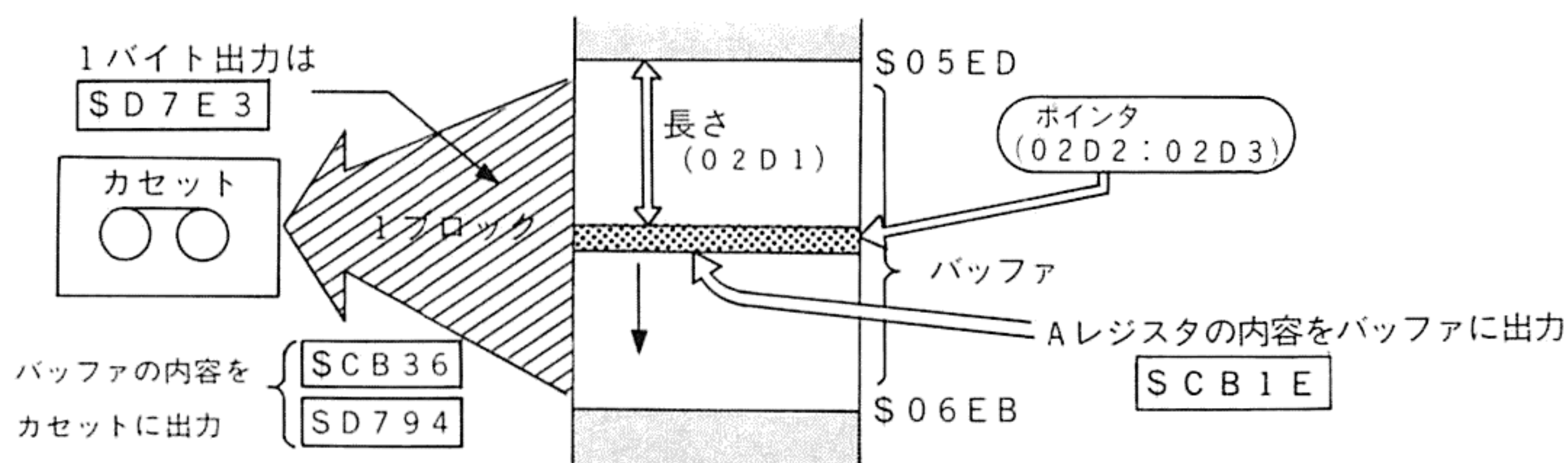
●次にエンドブロック（図4・4・3）を出力($CB38使用)し，その後モータをON（$D76F）して，テープに約1～2秒の空白領域を作ります．

入力モードと出力モードで共通に次の動作を行います．

●モータをOFF（$D772）し，カセット使用フラグ（02D0）をクリアして，カセットを使用しているファイルがなくなったことを示します．

4—4—4　カセット出力ルーチン

カセット出力ルーチンは4つありますが,その各々を上位ルーチンから下位ルーチンへと順番に並べると，**$CB1E**（バッファに1バイト出力)，**$CB36**（1ブロック出力上位ルーチン)，**$D794**（1ブロック出力下位ルーチン)，**$D7E3**（1バイトを直接カセットへ出力）というようになっています．

図4・4・5　　カセット出力ルーチン

(1)　カセット1バイト出力（バッファに）ルーチン

アドレス　$CB1E～$CB35

機　能　カセット（バッファ）に1バイト出力する．

レジスタ　A，B，X

入力情報　Aレジスタ＝出力データ

WORK　(02D1)　Ⓡ／Ⓦ　，(02D2：02D3)　Ⓡ／Ⓦ

SUB　$CB36→バッファからカセットへ1ブロック出力する

解　説　ポインタの示すバッファ内のアドレスにデータを１バイト出力しますが，バッファがいっぱいの場合は，バッファの内容をデータブロックとしてカセットに事前に出力します．これをワークエリアとの対応で見ると次のようになります．

●バッファ内のデータの長さ（０２Ｄ１）が２５５（＝＄ＦＦ），即ち，バッファ領域がいっぱいのときは，バッファの内容をカセットに出力します（＄ＣＢ３６）．

※出力した後は，（０２Ｄ１）は０に，（０２Ｄ２：０２Ｄ３）はバッファの先頭アドレス（＝＄０５ＥＤ）に設定されます．

●ポインタ（０２Ｄ２：０２Ｄ３）の指すバッファ内のアドレスに，Ａレジスタの内容を格納した後，ポインタ（０２Ｄ２：０２Ｄ３）を＋１します．同時にデータの長さ（０２Ｄ１）も＋１されます．

（２）　カセット１ブロック出力（バッファより）ルーチン

アドレス　＄ＣＢ３６，＄ＣＢ３８～＄ＣＢ４Ａ

機　能　バッファに格納されているデータを，データブロックとして（＄ＣＢ３６），あるいはその他のタイプのブロックとして（＄ＣＢ３８），それぞれカセットに出力する．

レジスタ　Ａ，Ｂ，Ｘ

入力情報　＄ＣＢ３８エントリの場合は，Ｂレジスタにブロックタイプを代入しておく（ヘッド…＄００，データ…＄０１，エンド…＄ＦＦ）．

ＷＯＲＫ　（０２Ｂ０：０２Ｂ１）Ⓡ，（０２ＣＣ：０２ＣＤ）Ⓦ
（０２ＣＥ）Ⓦ，（０２ＣＦ）Ⓦ
（０２Ｄ１）Ⓡ

ＳＵＢ　＄ＣＡ７０→バッファポインタ，データ数の初期設定
＄Ｄ７９４→カセット１ブロック出力ルーチン

解　説　＄ＣＢ３６をエントリとしてこのルーチンを呼んだ場合，（０２ＣＥ）に＄０１を代入しているので，データブロック（図４・４・４）として出力されます．他のタイプのブロックとして出力する場合は，＄ＣＢ３８をエントリとして，入力情報で示したようなパラメータをＢレジスタに代入しておくと，それが（０２ＣＥ）に代入されます．また，この後は次のような動作を行います．

●入出力先頭アドレス（０２ＣＣ：０２ＣＤ）にバッファ先頭アドレス＄０５ＥＤ（０２Ｂ０：０２Ｂ１）を代入し，今回出力するブロックのバイト数（０２ＣＦ）として，バッファ中のバイト数（０２Ｄ１）を設定します．
●＄Ｄ７９４を呼んで，バッファの内容を１ブロックとしてテープに出力した後，＄ＣＡ７０を使い，バッファのポインタ，データの長さの初期処理を行います．

（３）　カセット１ブロック出力（汎用）ルーチン

アドレス	＄Ｄ７９４～＄Ｄ７Ｅ２
機　能	カセットに１ブロック出力する．
レジスタ	Ａ，Ｂ，Ｘ
入力情報	（０２ＣＣ：０２ＣＤ）＝出力するメモリ領域の先頭アドレス （０２ＣＥ）＝出力するブロックのタイプ （０２ＣＦ）＝出力するブロックの長さ（バイト数）
ＷＯＲＫ	（０１Ｅ９）Ⓡ，（０１ＥＡ）Ⓡ （０２ＣＣ：０２ＣＤ）Ⓡ，（０２ＣＥ）Ⓡ （０２ＣＦ）Ⓡ，（０２Ｄ６）Ⓡ （０２Ｄ７）Ⓡ
ＳＵＢ	＄Ｄ７６Ｆ，＄Ｄ７７２→モータのＯＮ，ＯＦＦルーチン ＄Ｄ７８Ｃ→ギャップ出力ルーチン ＄Ｄ７Ｅ３→カセット１バイト出力ルーチン

解　説　動作の手順は次のようになります．
●モータをＯＮ（＄Ｄ７６Ｆ）します．モータをＯＮする前にすでにモータがＯＮ状態なら（０１Ｅ９）の内容（＝＄０Ａ）のバイト数，ＯＦＦ状態なら（０１ＥＡ）の内容（＝＄ＦＦ）のバイト数だけギャップを出力します．
●１ブロック出力します（図４・４・１）．
●モータスイッチフラグが立っていれば〔（０２Ｄ６）＝＄ＦＦ〕，モータをＯＦＦ（＄Ｄ７７２）します（この部分は，他のサブルーチンからも呼ばれており，そのエントリは＄Ｄ７ＤＤです）．
※ユーザがバッファを介してデータを出力する場合には，＄ＣＢ３６を使った方がよいのですが，バッファを介さないでメモリの内容を出力させたい場合には，このルーチンを使った方が有効となります．

(4)　カセット1バイト出力(直接カセットに)ルーチン

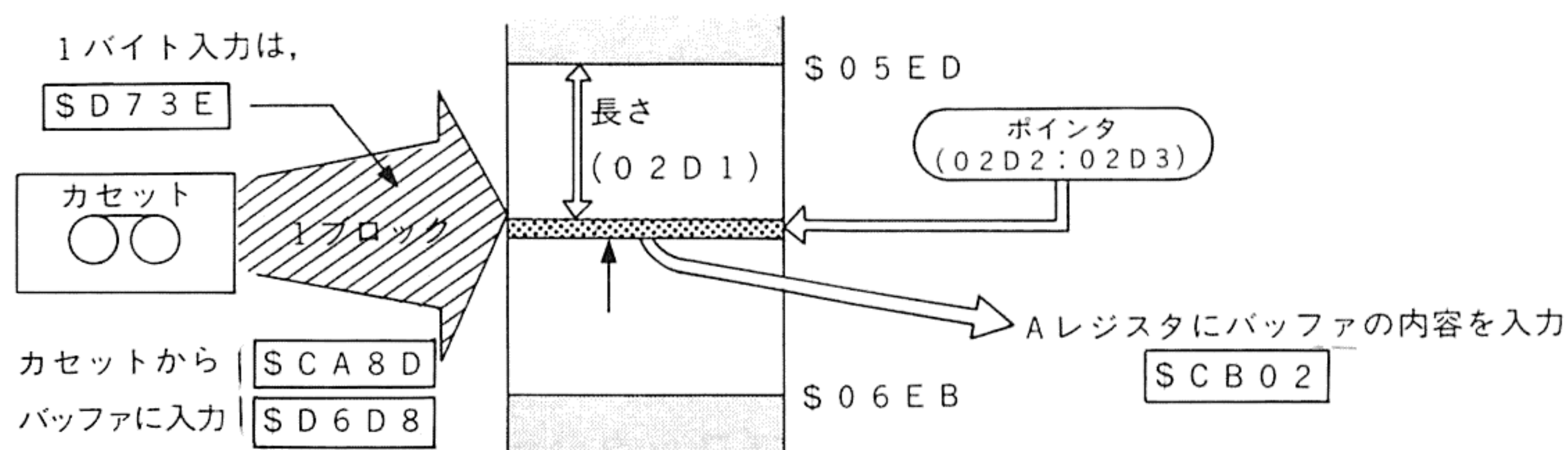
アドレス	$D7E3〜$D7F6
機　能	カセットにAレジスタの内容を直接出力する.
レジスタ	A, B, X, Y　(全て保存)
入力情報	Aレジスタ=出力するデータ
復帰情報	Cフラグ　=エラー発生時にセットされる
WORK	(05A0〜05A2)=BIOSへのRCB用　Ⓦ
SUB	$00DE→BIOSジャンプルーチン
	$D678→Abort ルーチン

解　説　BIOSのCTBWRT〔システム仕様2.1.4(2)〕を用いてカセットにAレジスタの内容を出力していますが，ブレーク・キーのチェックを行っているので，ブレーク・キーを押した場合は Abort します.

4-4-5　カセット入力ルーチン

カセット入力ルーチンも，出力ルーチンと同様に，4つのサブルーチンがあり，各々が出力の場合と同じレベルで対応しています. これらは，上位ルーチンから順番に$CB02（バッファから1バイト入力），$CA8D（バッファに1ブロック入力），$D6D8（1ブロック入力），$D73E（カセットから1バイト入力）となっています.

図4・4・6　カセット入力ルーチン

1バイト入力は，
$D73E
カセット
1ブロック
カセットから　$CA8D
バッファに入力　$D6D8
長さ
(02D1)
$05ED
$06EB
ポインタ
(02D2:02D3)
Aレジスタにバッファの内容を入力
$CB02

(1)　カセット1バイト入力(バッファから)ルーチン

アドレス	$CB02～$CB1D
機　能	カセット(バッファ)から1バイト入力する.
入力情報	A, B, X
復帰情報	Aレジスタ　＝入力データ (00C0)＝入力するデータがないと$FFに設定される.
WORK	(02D1)　Ⓡ／Ⓦ　,(02D2：02D3)　Ⓡ／Ⓦ (00C0)＝入力データ終了フラグ　Ⓦ
SUB	$CA8D→カセットからバッファへ1ブロック入力

解　説　このルーチンの動作内容は，次のようになります.

●データがない〔(02D1)＝0〕ときは，(00C0)を反転させて$FFとし，リターンします(終了判断).

●それ以外のときは，ポインタ(02D2：02D3)の指すバッファ(05ED～06EB)内のアドレスからAレジスタに1バイト読み込んだ後，ポインタ(02D2：02D3)を＋1し，データの長さ(02D1)を－1します.

●このとき，バッファ内にデータがなくなったら〔(02D1)＝0〕，次のブロックをバッファに読み込みます($CA8D).

※ここで，(02D1)と(02D2：02D3)は読み込んだブロックがデータブロックである場合には新たに初期設定されるのに対し，エンドブロックである場合には(02D1)＝0とし，次にこのルーチンを呼び出したときにファイルの読み込みが既に終了していることが認知できるようにします.

(2)　カセット1ブロック(データブロック，エンドブロック)入力ルーチン

アドレス	$CA8D～$CAAD
機　能	カセットからバッファに1ブロック(データ／エンド)入力する.
レジスタ	A, B, X
WORK	(02B1：02B2)　Ⓡ　,(02CC：02CD)　Ⓦ (02CE)　Ⓡ　,(02CF)　Ⓡ

（０２Ｄ１）　Ⓦ　　　　　，（０２Ｄ２：０２Ｄ３）　Ⓦ

ＳＵＢ	＄Ｄ６Ｄ８→カセット１ブロック入力ルーチン
終了条件	エラーの際は，"Device I／O Error"（＄ＣＡＢ１）を発生させる．

解　説　ワークエリアと関連づけて，次の動作が行われます．

●入出力先頭アドレス（０２ＣＣ：０２ＣＤ）にバッファの先頭アドレス＄０５ＥＤ（０２Ｂ０：０２Ｂ１）を代入し，＄Ｄ６Ｄ８を使い１ブロックをバッファに取り込みます．このとき，エラーが起こった場合や，ヘッダブロックを読み込んだ場合〔(０２ＣＥ)＝＄００〕は，＄ＣＡＢ１にジャンプします．また，エンドブロック〔(０２ＣＥ)＝＄ＦＦ〕のときは，この時点でリターンします．

●ブロックの長さ（０２ＣＦ）を見て，０である場合は始めからやり直します．それ以外のときは，ブロックの長さをバッファ中のデータ数（０２Ｄ１）に代入し，バッファの先頭アドレス＄０５ＥＤをバッファポインタ（０２Ｄ２：０２Ｄ３）に代入します．

（３）　カセット１ブロック入力（汎用）ルーチン

アドレス	＄Ｄ６Ｄ８～＄Ｄ７２Ｆ
機　能	カセットから１ブロック入力する．
レジスタ	Ａ，Ｂ，Ｘ
入力情報	（０２ＣＣ：０２ＣＤ）＝入力先頭アドレス
復帰情報	（０２ＣＥ：０２ＣＦ）＝ブロックの（タイプ：長さ）
	Ｂレジスタ　＝チェックサムの値
	Ｚフラグ　＝エラーがないときにセットされる．
ＷＯＲＫ	（０２ＣＣ：０２ＣＤ）Ⓡ，（０２ＣＥ）Ⓦ
	（０２ＣＦ）Ⓦ，（０２Ｄ５）Ⓡ／Ⓦ
ＳＵＢ	＄Ｄ７３０→エラーがなくなるまで，１バイト読み込む
	＄Ｄ７３５→１バイト読む．エラーの際はこのルーチンを終わらせる．
	＄Ｄ７６Ｆ→モータＯＮルーチン
	＄Ｄ７ＤＤ→（０２Ｄ６）≠０の場合は，モータをＯＦＦする．

解　説　任意のブロックを読み込める，このルーチンは次の動作をします．

●５バイト以上のギャップ(＝＄ＦＦ)を確認した後，＄０１，＄３Ｃのブロック識別子を確認し，(０２ＣＥ)にブロックタイプ，(０２ＣＦ)にブロックの長さをカセットから読んで,，代入します（＄Ｄ７３０と＄Ｄ７３５を使用)．
●この後，(０２ＣＣ：０２ＣＤ)の示す先頭番地から，(０２ＣＦ)の示すバイト数だけデータを読み込みます(＄Ｄ７３５使用)．この間，チェックサムを計算し，最後に読み込むチェックサムと照合し，違っていれば(０２Ｄ５)に＄０１を代入します．終わりに＄Ｄ７ＤＤを使い，モータのＯＦＦを決定します．

(４)　１バイト入力（直接カセットから）ルーチン

アドレス	＄Ｄ７３Ｅ～＄Ｄ７５７　（＄Ｄ７３０～＄Ｄ７３４)
機能	カセットからＡレジスタへ１バイト入力する
レジスタ	Ａ，Ｘ　(Ｂ，Ｘ，Ｙは保存)
復帰情報	Ａレジスタ＝カセットからの入力データ Ｃフラグ　＝エラーの場合，セットされる
ＷＯＲＫ	(０５Ａ０～０５Ａ２)＝ＢＩＯＳへのＲＣＢ　Ⓡ／Ⓦ
ＳＵＢ	＄Ｄ６７８→Abort ルーチン ＄００ＤＥ→ＢＩＯＳジャンプルーチン

解説　ＢＩＯＳのＣＴＢＲＥＤ〔システム仕様２．１．４(３)〕を用いています．＄Ｄ７３０はエラーがなくなるまで読み込みをするというものです．

４—４—６　その他の主なカセット関係ルーチンの概要

(１)　モータのＯＮ・ＯＦＦ

アドレス	＄Ｄ７５８～＄Ｄ７８Ｂ
レジスタ	Ａ，Ｂ，Ｘ　(Ｘは保存)

解説　ＯＮするのは＄Ｄ７６Ｆ，＄Ｄ７７８，ＯＦＦするのは＄Ｄ７７２，＄Ｄ７７Ｂというエントリがあります．＄Ｄ７７８，＄Ｄ７７Ｂはモータの状態フラグ（０２Ｄ７）を変化させません（このルーチンはＢＩＯＳのＭＯＴＯＲを用います)．

（2） ヘッダブロックを読み込んで，ファイル名を画面に出力

アドレス　＄ＣＢ５７～＄ＣＢＢ４

解　説　ファイルのヘッダブロックを入力し，ファイル名の指定があれば，その照合を行います．ＢＡＳＩＣのプログラム中でなければ，"ＦＯＵＮＤ：" ＋ファイル名や "ＳＫＩＰ：" などを表示し，指定のファイルと異なるときは，再びファイルのヘッダの読み込みを繰り返します．

４—４—７　サンプル

カセットから１ブロック読み込み，その内容を１６進数で表示させるというサンプルです．注意すべきことは，**バイナリセーブの場合，ブロック間のギャップ数が短いため，続けて読むときにブロックを１つ読み飛ばす**ということです．これを防ぐには，**少し巻き戻しておくか，セーブする際にあらかじめ（０１Ｅ９）の ギャップ数を＄８０**ぐらいに増やしておくことが必要です．なお，このルーチンではデータバッファとして＄４０００～のメモリ空間を使用しております．

```
PAGE  001 (821201,000732) CASSET

01000                          * Load 1 Block from Casset
01010                          * and Print out it
01020                                   NAM .   CASSET
01030                                   OPT     MEM,NOGEN
01040              0000                 SETDP   $00
01050  5000                             ORG     $5000
01060  5000 BD     D76F                 JSR     $D76F     * MOTOR ON
01070  5003 5F                          CLRB
01080  5004 34     04                   PSHS    B
01090  5006 8E     4000                 LDX     #$4000
01100              5009         REP     EQU     *
01110  5009 BD     D730                 JSR     $D730     * $01 Check
01120  500C 81     01                   CMPA    #$01
01130  500E 26     F9    5009           BNE     REP
01140  5010 BD     D730                 JSR     $D730     * $3C Check
01150  5013 81     3C                   CMPA    #$3C
01160  5015 26     F2    5009           BNE     REP
01170  5017 BD     D730                 JSR     $D730     * Block Type Input
01180  501A A7     80                   STA     ,X+
01190  501C BD     D730                 JSR     $D730     * Block Length Input
01200  501F A7     80                   STA     ,X+
01210  5021 A7     E4                   STA     ,S
01220              5023         REP2    EQU     *
01230  5023 27     09    502E           BEQ     ESC1      * If Length = 0 then esc1
01240  5025 BD     D730                 JSR     $D730     * Data Input
01250  5028 A7     80                   STA     ,X+
01260  502A 6A     E4                   DEC     ,S
01270  502C 20     F5    5023           BRA     REP2
01280              502E         ESC1    EQU     *
01290  502E BD     D730                 JSR     $D730     * Check Sum Input
01300  5031 A7     84                   STA     ,X
01310  5033 BD     D772                 JSR     $D772     * MOTOR OFF
01320  5036 35     04                   PULS    B
01330  5038 0F     BF                   CLR     $BF
01340  503A 30     8D 0057              LEAX    MES1-1,PCR
```

```
01350  503E BD    9BDB                JSR    $9BDB    * Message1 Printout
01360  5041 BD    9B50                JSR    $9B50    * Line Feed
01370  5044 108E  4000                LDY    #$4000
01380  5048 30    8D 0055             LEAX   MES2-1,PCR
01390  504C BD    9BDB                JSR    $9BDB    * Message2 Printout
01400  504F A6    A0                  LDA    ,Y+      * Block Type Printout
01410  5051 BD    AC3D                JSR    $AC3D    * Hex$ Printout (A register)
01420  5054 BD    9B50                JSR    $9B50    * Line Feed
01430  5057 30    8D 0072             LEAX   MES3-1,PCR
01440  505B BD    9BDB                JSR    $9BDB    * Message3 Printout
01450  505E A6    A0                  LDA    ,Y+      * Block Length Printout
01460  5060 34    02                  PSHS   A
01470  5062 BD    AC3D                JSR    $AC3D    * Hex$ Printout
01480  5065 30    8D 0076             LEAX   MES4-1,PCR
01490  5069 BD    9BDB                JSR    $9BDB    * Message4 Printout
01500  506C BD    9B50                JSR    $9B50    * Line Feed
01510  506F 6D    E4                  TST    ,S
01520  5071 27    12    5085          BEQ    ESC2     * If Length = 0 then esc2
01530             5073         REP3   EQU    *
01540  5073 A6    A0                  LDA    ,Y+      * Data Printout
01550  5075 BD    AC3D                JSR    $AC3D    * Hex$ Printout
01560  5078 BD    9C22                JSR    $9C22    * Space Printout
01570  507B BD    9C22                JSR    $9C22
01580  507E 6A    E4                  DEC    ,S
01590  5080 26    F1    5073          BNE    REP3
01600  5082 BD    9B50                JSR    $9B50    * Line Feed
01610             5085         ESC2   EQU    *
01620  5085 30    8D 005D             LEAX   MES5-1,PCR
01630  5089 BD    9BDB                JSR    $9BDB    * Messae5 Printout
01640  508C A6    A4                  LDA    ,Y       * Check Sum Printout
01650  508E BD    AC3D                JSR    $AC3D    * Hex$ Printuot
01660  5091 BD    9B50                JSR    $9B50    * Line Feed
01670  5094 35    84                  PULS   PC,B
01680             5096         MES1   EQU    *
01690  5096       31                  FCC    !1 BLOCK DMP!
01700  50A1       00                  FCB    $00
01710             50A2         MES2   EQU    *
01720  50A2       42                  FCC    !BLOCK TYPE ($00=HEAD!
01730  50B6       20                  FCC    ! $01=DATA $FF=END) ·· $!
01740  50CD       00                  FCB    $00
01750             50CE         MES3   EQU    *
01760  50CE       42                  FCC    !BLOCK LENGTH ·· $!
01770  50DF       00                  FCB    $00
01780             50E0         MES4   EQU    *
01790  50E0       20                  FCC    ! Bytes!
01800  50E6       00                  FCB    $00
01810             50E7         MES5   EQU    *
01820  50E7       43                  FCC    !CHECK SUM ·· $!
01830  50F5       00                  FCB    $00
01840             5000                END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=50F5
PROGRAM ENTRY ADDR=5000


EXEC&H5000

1 BLOCK DMP
BLOCK TYPE ($00=HEAD $01=DATA $FF=END) ·· $00
BLOCK LENGTH ·· $14 Bytes
56  49  50  2E  46  4D  2D  37  00  00  00  00  00  00  00  00  00  00  00  00
CHECK SUM ·· $28

Ready
```

4－5　ディスク入出力ルーチン

4－5－1　ディスク入出力ルーチンの概要とワークエリア

ディスク入出力ルーチンは，ＦＡＴとディレクトリスロットからファイルの位置を割り出しています．この際，クラスタ番号（０～１５１）とクラスタ内のセクタ番号（０～７）という論理的な番号から，トラック番号とセクタ番号に変換しています．ただし，変換のしやすさから，セクタ番号の値は－１されています（セクタ番号は０～３１の値をとります）．なお，名称やその内容については，４－１節や文法書の２．１０．３を御参照下さい．また，本節では下図に示したため，ワークエリアは省略しました．

図４・５・１　ディスク入出力ルーチンのワークエリア

アドレス	名称および機能
７１Ｄ６～７１Ｄ７	ドライブテーブルの先頭アドレス；ドライブテーブルが複数あるときはこの値から計算されます（１－４－４(２)および表１・４・１～２参照）．
７１Ｄ８～７１Ｄ９	システムランダムバッファの先頭アドレス；ＤＳＫＩ＄，ＤＳＫＯ＄で使用されるファイルバッファです（１－４－４(３)および表１・４・１～２参照）．
７１ＤＡ～７１Ｆ９	ファイルバッファ＃＃０～＃＃１５の先頭アドレス；２バイトずつ入ります．ただし，電源投入時に入力されたファイルバッファ数より大きい番号のものは＄００００になっています（１－４－４(３)および表１・４・１～２参照）．
７１ＦＡ	ドライブ数－１（０～３）；Ｆ－ＢＡＳＩＣではドライブ番号が０から始まるので１つ少ない値になっています．
７１ＦＢ	ファイルバッファ数（０～１５）；ファイルバッファが＃＃０から始まるので１つ少ない値になっています．
７１ＦＣ～７１ＦＤ	セクタ入出力の先頭アドレス；セクタと入出力を行う際のデータの先頭アドレスが入っています．
７１ＦＥ	ドライブ番号；入出力の際のドライブ番号を入れておきます．
７１ＦＦ	入出力機能；＄７４ＣＥ，＄７４Ｅ６を呼び出す際に，ＢＩＯＳに処理させる入出力の機能を次のように与えます．＄００－ＲＥＳＴＯＲＥ，＄０１－何も処理を行わない，＄０２－ＤＲＥＡＤ，＄０３－ＤＷＲＩＴＥ
７２００	セクタ番号－１；入出力の際の実際のセクタ番号より１つ少ない値を入れておきます．
７２０１	エラーステータス；入出力の際にＢＩＯＳが与えるエラー番号が入ります．
７２０２	トラック番号；入出力の際のトラック番号を入れておきます．
７２０３～７２０Ａ	ＢＩＯＳを呼び出すためのＲＣＢ
７２０Ｂ	目的のファイル名（ディレクトリスロット）のあるトラック１内のセクタ番号－１
７２０Ｃ～７２０Ｄ	目的のファイルのディレクトリスロットのあるセクタを汎用ディスクバッファに読み込んだときの，ディレクトリスロットのバッファ内の先頭アドレス
７２０Ｅ	目的のファイルの先頭クラスタ番号
７２０Ｆ	目的のファイルのランダムアクセスフラグ
７２１０	空きディレクトリスロットのあるトラック１内のセクタ番号－１
７２１１～７２１２	空きディレクトリスロットのあるセクタを汎用ディスクバッファに読み込んだときの，ディレクトリスロットのバッファ内の先頭アドレス
７２１３～７３１２	汎用ディスクバッファ
７３１３	現在アクセス中のファイルバッファの番号

4—5—2　ディスク　オープン（ＯＰＥＮ）ルーチン

アドレス　＄７７１３〜＄７８３Ａ

機　能　ディスクに割り当てられているファイルをオープンする．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　Ｂレジスタ　＝ファイルの入出力モードのモード文字（“Ｉ”，“Ｏ”，“Ａ”，“Ｒ”）のアスキーコード

（００ＢＦ）＝ファイル番号

（０２ＤＢ）＝ファイルタイプ

（０２ＤＣ）＝アスキーフラグ

（０２ＤＤ）＝ファイル名の長さ

（０２ＤＥ〜０２Ｅ５）＝ファイル名（アスキーコード）

（０２Ｅ６）＝デバイス番号（≧＄８０）

復帰情報　（０２ＤＡ）＝ファイルのモード

そのファイル番号のＳＦＤ（システムファイルディスクリプタ）

ＷＯＲＫ　省略（図４・５・１参照）

ＳＵＢ　＄７４ＡＢ→ファイルバッファの先頭アドレスの頭出し

＄７４ＢＤ→ドライブテーブルの先頭アドレスの頭出し

＄７４ＣＥ→セクタ入出力ルーチン

＄７５Ｄ４→「ＫＩＬＬ」ルーチンの実行部

＄７６２Ｃ→ディレクトリ探索ルーチン

＄７６８Ｅ→ファイルがディスクにあるかどうかのチェック

＄７６９８→ファイルがすでにオープンされているかどうかのチェック

＄７６ＢＦ→ＩＤ確認とＦＡＴのドライブテーブルへの読み込み

＄７８３Ｂ→ファイルバッファの初期化ルーチン

＄７８５Ｆ→ディレクトリスロット，ＦＡＴの記入ルーチン

＄７９６９→次セクタ設定ルーチン

＄７９ＡＡ→「クラスタ，セクタ➡トラック，セクタ」変換ルーチン

＄７Ａ３７→次セクタ読み込みルーチン

＄７Ａ８６→ポインタをＥＯＦまで進める

終了条件　エラーのあるときは，エラーコードをＢレジスタに入れてエラー処理ルーチン（＄８ＤＤ１）にジャンプする．

図4・5・2　ディスクオープンルーチンの動作

$7713
プログラムファイル?　YES: ファイルバッファ＃0
NO
空きＦＢＵＦを探し，あればそのＦＢＵＦを割り当てる　空きＦＢＵＦがないとき ①
ＩＤチェック，ＦＡＴをドライブテーブルに読み込む　76BF
ディレクトリを探索し，同名のファイルがあるか調査する　762C
モード

"I"　7745
ファイルモード＝$10
ファイル存在チェック　768E
ファイルオープンチェック　7698
ランダムアクセスフラグ　ランダム ②
ファイルタイプとアスキーフラグの設定
ＦＢＵＦの初期化　783B
ファイルの先頭セクタをＦＢＵＦに読み込む　7A37
7763

"O"　7786
ファイルモード＝$20
ファイルがある?　NO
YES
プログラムファイル?　NO
YES
ＢＡＳＩＣのプログラム上?　YES ③
NO
ＫＩＬＬ古いファイル　75D4
古いファイルのあったディレクトリの位置を記憶
ディレクトリの記入，ＦＡＴの記入　785F
ＦＢＵＦの初期化　783B
77B6

"A"　77BE
ファイルモード＝$20
ファイル存在チェック　768E
ファイルオープンチェック　7698
ランダムアクセスフラグ　ランダム ②
ＦＢＵＦの初期設定　783B
最終クラスタ探し
ＦＣＢに最終クラスタ，クラスタ内のセクタ番号設定
トラック，セクタ番号変換　79AA
ＦＢＵＦにＤＲＥＡＤ　74CE
ＤＷＲＩＴＥ　74CE
ポインタをＥＯＦまで進める　7A86
バッファ内が一杯か?　NO
YES
次のセクタを設定　7969
781C

"R"　7822　②
ファイルモード＝$40
ファイルがある?　YES
NO
ディレクトリの記入　ＦＡＴの記入　785F
ランダムアクセスフラグ　②
ランダム
ＦＢＵＦの初期化　783B
レコード番号＝1
783A

7764
ドライブテーブルのアクセス数を＋1する
ＳＦＤの設定
$777D

他のエラーについては，サブルーチン内部で発生すれば，そこからエラー処理ルーチンへジャンプする．

①…"Too Many Open Disk Files"
②…"Bad File Mode"
③…"File Already Exists"

解　説　このルーチンの動作を図4・5・2に基づいて説明します.

● プログラムファイル〔(00BF)＝＄11〕ならFBUF（ファイルバッファ）＃＃0を割り当てますが，データファイルの場合は空きFBUFを探し（FBUF番号の大きいものから順に），未使用のFBUFがあればそのFBUFに割り当てます.
● IDセクタのチェックをし，まだ読み込んでなければ，ディスクからFATをドライブテーブルに読み込みます.
● ディレクトリを探索し，入力情報で与えられたファイル名と同じ名をもつファイルがあるか調査し，あれば（720B～720F）を設定します.
● この後は各モードに分かれ，そのモードの動作を行います．注意すべきことは，"I"と"A"のモードで，ファイルの存在チェックというのはディレクトリ探索が既に実行されているので，（720B)が設定してあるかどうかを判別するだけで再度ディレクトリ探索をしているのではないということと，ファイルオープンチェックを呼び出す際に，"I"の場合は同じモードであれば複数個のファイル番号で，同一のファイルを開くことが許されるのに対して，"A"の場合にはそれが許可されないということです.
● 各モードの処理が終わった後は，共通してドライブテーブルのアクセス数（4－1－3節参照）を＋1して，ファイルモードとFBUFの番号から，そのファイル番号の示すSFDを設定しています.

4―5―3　ディスク　クローズ（CLOSE）ルーチン

アドレス	＄78BE～＄7913
機　能	ディスクに割り当てられているファイルをクローズする.
レジスタ	A，B，X，U
入力情報	Aレジスタ＝そのファイルのSFD Bレジスタ＝ファイル番号
WORK	（00BF）Ⓡ　　，（02B4：02B5）Ⓡ その他は省略（図4・5・1）
SUB	＄74AB→ファイルバッファの頭出し ＄74BD→ドライブテーブルの頭出し ＄74CE→セクタ入出力ルーチン ＄75FB→FAT記入ルーチン

＄７９１４→ＳＦＤからドライブ番号を割り出す
＄７９ＡＡ→クラスタ，セクタ→トラック，セクタ変換ルーチン

解　説　次のような手順でファイルをクローズしていきます．

- ドライブテーブルの内容がＦＡＴに記入されていない場合，記入を行います．
- 出力モードで，まだディスクに未転送のデータがバッファに残されている場合，それを出力します．このとき，データ終了を示す識別子として＄１Ａが出力され，そこからバッファの終わりまでをクリアして出力されます（ただし，バッファの終わりでデータが終わっているときは出力するのみで，これらの処理は行いません）．
- ドライブテーブルのファイルのアクセス数を－１し，今まで使用していたＦＢＵＦの先頭に＄００を代入して使用されなくなったことを示します．そして，そのファイルのＳＦＤを＄００にします．

４―５―４　入出力ルーチン

（１）　ディスク　１バイト出力ルーチン

アドレス	＄７９２４，＄７９６９～＄７９８Ｅ
機　能	ＦＢＵＦに１バイトのデータを出力する．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報	（Ｓ～Ｓ＋７）＝リターン時のＡ，Ｂ，Ｘ，Ｙ，Ｕレジスタの値 （Ｓ＋８：Ｓ＋９）＝リターンアドレス Ａレジスタ＝出力データ（１バイト） Ｂレジスタ＝そのファイルのＳＦＤの内容
ＳＵＢ	＄７４ＢＤ→ドライブテーブルの頭出し ＄７４ＣＥ→セクタ入出力ルーチン ＄７８Ａ９→ＦＡＴ記入ルーチン ＄７９９０→空きクラスタ検索・設定ルーチン ＄７９ＡＡ→「クラスタ，セクタ➡トラック，セクタ」変換ルーチン

解　説　次のような処理を行います．

- Ｂレジスタの値から，そのファイルが使用中のファイルバッファを選択します．
- 出力コードが＄０Ｄ（ＣＲ）の場合（ＦＢＵＦ＋６；データのレコード長）を

0にします．また，出力コードが＄２０以上の場合は，(ＦＢＵＦ＋６）を＋１してレコード長が増えたことを示します．

● （ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）の値を＋１して，それが＄０１００以下ならば，Ｉ／Ｏバッファのそのポインタの示す箇所にＡレジスタの内容を格納します．＄０１００になった場合は，現在アクセス中のセクタにＩ／Ｏバッファの内容を出力します．

● ＄０１００である場合は，この後＄７９６９以降で次のセクタをアクセスするためのＦＣＢの設定を行います．

● （ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）のポインタの値を0にします．

● 次のセクタを（ＦＢＵＦ＋４）に設定します〔ただし，クラスタ内にセクタが残っていない場合は，ＤＲＶＴＢＬで空きクラスタを検索して，次のセクタを空きクラスタの先頭セクタに割り当てるように(ＦＢＵＦ＋３)を設定します〕．

● ＦＡＴに記入します．

（２） ディスク　１バイト入力ルーチン

アドレス	＄７９Ｆ９～＄７Ａ３５
機　能	ＦＢＵＦから１バイトのデータを入力する．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報	（Ｓ～Ｓ＋６）＝リターン時のＢ，Ｘ，Ｙ，Ｕレジスタの値 （Ｓ＋７：Ｓ＋８）＝リターンアドレス Ｂレジスタ＝そのファイルのＳＦＤの内容
復帰情報	Ａレジスタ＝入力データ （００Ｃ０）＝データ終了時に＄ＦＦになる．
ＳＵＢ	＄７Ａ３７→次セクタ読み込みルーチン

解　説　場合によって動作が異なります．

● レコードが途中で切れている場合〔(ＦＢＵＦ＋１１)＝＄ＦＦ〕は，蓄えておいた入力データ（ＦＢＵＦ＋１２）の内容をＡレジスタに与えます．

● 入力するデータがない場合〔(ＦＢＵＦ＋１３：ＦＢＵＦ＋１４)＝0〕は，（００Ｃ０）に＄ＦＦを代入してリターンします．

● 入力データがある場合は，ポインタ（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）の内容を－１し，データポインタ（ＦＢＵＦ＋５）の指すデータの内容をＡレジス

タに代入して，データポインタを＋１します．この際，Ｉ／Ｏバッファにデータがなくなると，データポインタ（ＦＢＵＦ＋５）をクリアし，次セクタをＩ／Ｏバッファに読み込みます〔同時に，(ＦＢＵＦ＋１３：ＦＢＵＦ＋１４)が再設定されます．ただし，エラーやファイルの終了時などは再設定されません〕．

（３）　次セクタ読み込みルーチン

アドレス　＄７Ａ３７，＄７Ａ８６～＄７ＡＡ７

機　能　データを入力するために，Ｉ／Ｏバッファにデータがなくなった際に次のセクタから，データをＩ／Ｏバッファに読み込む．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ

入力情報　Ｘレジスタ＝ＦＢＵＦの先頭アドレス

復帰情報　Ｃフラグ　＝ファイルの最後のクラスタの最終セクタで次のセクタを読み込もうとしたときセットされる．

ＳＵＢ　＄７４ＢＤ→ドライブテーブルの頭出しルーチン
＄７４ＣＥ→セクタ入出力ルーチン
＄７９ＡＡ→「クラスタ，セクタ➡トラック，セクタ」変換ルーチン
＄７Ａ９２→ポインタの初期設定処理
＄７ＡＡ０→ポインタのデクリメント（－１）処理

終了処理　ファイルの最終クラスタのセクタ数が８以上になっていた場合（０～７が普通）は，“Bad File Structure” エラー

解　説　アクセス中のクラスタ内セクタ番号（ＦＢＵＦ＋４）を＋１して，その番号のセクタを読み込みますが，その番号が８になっていたときは，現クラスタの継続クラスタの先頭セクタからデータを読み込みます．この際，そのクラスタが継続クラスタを持たない場合，あるいはファイルが使用している最終セクタでこのルーチンを呼び出した場合などは，Ｃフラグをセットして何もしないでリターンします．

次のセクタを読み込んだときは，ポインタ（ＦＢＵＦ＋１３：ＦＢＵＦ＋１４）を＄０１００に再設定します．しかし，読み込んだセクタが最終セクタの場合は，そのポインタを＄１Ａ（データ終了の識別子）の前までの値に設定しなければなりません．これを行うのが＄７Ａ８６以降です．

(4) セクタ入出力ルーチン

アドレス	$74CE～$74E5 ($74E6～$755C)
機　能	BIOSを用いてセクタの入出力,シークトラック0の処理を行う.
レジスタ	A, B, X (B, X, Y, Uは保存)
入力情報	(71FC:71FD)=セクタの入出力の先頭アドレス (71FE)=ドライブ番号(0～3) (71FF)=入出力機能 $00……RESTORE　$01……無動作 $02……DREAD　$03……DWRITE (7200)=セクタ番号−1(0～31) (7202)=トラック番号(0～39)
復帰情報	(7201)=BIOSの設定するエラー番号
SUB	$74E6→セクタ入出力実行ルーチン
終了条件	BIOSの設定するエラーにより"Drive Not Ready"エラー,"Disk Write Protected"エラー,"Drive I/O Error"を発生させる($8DD1).

解　説　このルーチンは,DISK BASICの根幹となる入出力ルーチンです.このルーチンの下位ルーチン$74E6は,(71FF)の値を参考に,BIOSのRESTORE,DREAD,DWRITEを呼び出すための処理をするルーチンを(755D～7564)のジャンプテーブルに従ってコールします.各々のルーチンは,(7203～720A)にRCBを設定してBIOSを呼び出します.

4—5—5 その他の主なディスク関係ルーチンの概要

(1) FAT初期化ルーチン

アドレス	$7422～$7444
レジスタ	A, B, U (Uは保存されるが,入力情報として使用)

解　説　(71FE)で指定するドライブのFATの初期化(システム領域以外のものは$FFで埋める)を行います.入力情報として他に(71FC:7

１ＦＤ）にＵレジスタの値－１のアドレスを与えることと，（７２０２）に＄０１を代入しておくことが必要で，このルーチン終了直後に，＄７４ＣＥを呼び出すことによって初期化された内容がＦＡＴに書き込まれることになります．

（２） ファイルバッファ（ＦＢＵＦ）の頭出しルーチン

アドレス ＄７４ＡＢ，＄７４Ｂ１～＄７４ＢＣ
レジスタ Ｂ，Ｘ （Ｂは保存されるが，入力情報として使用）

解　説 ＄７４ＡＢエントリの際は（７３１３）に，＄７４Ｂ１エントリの際はＢレジスタに，それぞれＦＢＵＦの番号（＄００～＄０Ｆ）を入力情報として与えると，（７１ＤＡ～７１Ｆ９）のワークエリアから，そのＦＢＵＦの先頭アドレスをＸレジスタに代入してきます．このとき，そのＦＢＵＦが未使用ならばＺフラグをセットして返してきます．

（３） ドライブテーブル（ＤＲＶＴＢＬ）の頭出しルーチン

アドレス ＄７４ＢＤ～＄７４ＣＤ
レジスタ Ａ，Ｂ，Ｘ （Ａ，Ｂは保存）

解　説 （７１ＦＥ）のドライブ番号および，（７１Ｄ６：７１Ｄ７）の先頭アドレスから，そのＤＲＶＴＢＬの先頭アドレスをＸレジスタに代入し，また，ＦＡＴをまだ読み込んでいないのなら，Ｚフラグをセットして返してきます．

（４） ＦＡＴ記入ルーチン

アドレス ＄７５ＦＢ～＄７６２Ｂ
レジスタ Ａ，Ｂ，Ｘ，Ｕ

解　説 （７１ＦＥ）で指定するドライブのＦＡＴにＤＲＶＴＢＬの内容を記入します．この際，汎用ディスクバッファ（７２１３～７３１２）を使用しています．また，記入後のＤＲＶＴＢＬの先頭アドレス＋１番地の内容をクリアして記入済であることを示します．

(5) ディレクトリ探索ルーチン

アドレス　$762C ～ $768D
レジスタ　A, B, X, Y, U

解　説　(71FE) で指定するドライブのディレクトリの探索を行い，(02DE～02E5) で指定したファイル名の探索，およびファイル名を特に指定しない場合は，空きディレクトリスロットの位置の探索をします（両者は別々にではなく同時に行われます．これは，新しいファイルを作る場合などに両方の情報が必要となってくるためです)．この探索によって次のワークエリアを設定します．

- (720B)　＝指定したファイル名があるセクタ番号－1（3～31)，ファイル名がない場合は0になります．
- (720C～720F)　＝ファイル名がある場合は図4・5・1の内容が入ります．
- (7210)　＝空きディレクトリスロットがあるセクタ番号－1（3～31)，スロットがない場合は0になります．
- (7211：7212)　＝スロットがある場合は図4・5・1の内容が入ります．

(6) ファイルのオープンチェックルーチン

アドレス　$7698～$76BE
レジスタ　A, B, X　(Aは保存されるが，入力情報として使用)

解　説　(71FE) が指すドライブの，同じファイルをアクセスしている他のファイルバッファの有無を調べます．入力情報としてAレジスタにアクセスしたいファイルのファイルモードを与えます．同じモードならエラーは起こりません（オープンルーチンの“A”のモードでこのルーチンを呼んでいるときは，Aレジスタに$FFを代入しているのでエラーになります)．このルーチンは，ディレクトリ探索ルーチンをあらかじめ呼び出してから使用します．

(7)　ＩＤ確認とＦＡＴ読み込みルーチン

アドレス　＄７６ＢＦ，＄７６Ｄ２～＄７７１２
レジスタ　Ａ，Ｂ，Ｘ，Ｕ

解　説　＄７６ＢＦエントリの際はデバイス番号（０２Ｅ６）の示すドライブから，＄７６Ｄ２エントリの際は（７１ＦＥ）の示すドライブから，それぞれＦＡＴをＤＲＶＴＢＬに読み込みます．その際にＩＤ確認を行いますが，そのドライブのＤＲＶＴＢＬをアクセスしているファイルがある場合は読み込みません．なお，このルーチンも汎用ディスクバッファを利用しています．

(8)　ファイルバッファの初期化ルーチン

アドレス　＄７８３Ｂ～＄７８５Ｅ
レジスタ　Ａ，Ｂ，Ｘ，Ｕ

解　説　（７３１３）の指すＦＢＵＦを初期化します．ＦＢＵＦの内容をすべてクリアした後，ＦＣＢの一部は次のように設定されます．

- （ＦＢＵＦ＋１）〔ドライブ番号〕＝（７１ＦＥ）
- （ＦＢＵＦ＋２）〔ファイル先頭クラスタ番号〕＝（７２０Ｅ）
- （ＦＢＵＦ＋３）〔アクセス中のクラスタ番号〕＝（７２０Ｅ）
- （ＦＢＵＦ＋７：ＦＢＵＦ＋８）〔Ｉ／Ｏバッファ長〕＝（＄０１００）
- （ＦＢＵＦ＋９：ＦＢＵＦ＋１０）〔Ｉ／Ｏバッファの先頭アドレス〕
  ＝ＦＢＵＦの先頭アドレス＋１５

(9)　ディレクトリスロット記入とＦＡＴ記入ルーチン

アドレス　＄７８５Ｆ，＄７８Ａ９～＄７８Ｂ９
レジスタ　Ａ，Ｂ，Ｘ，Ｕ

解　説　このルーチンもディレクトリ探索ルーチンを事前に実行する必要があり，（７２１０）が示すセクタを汎用ディスクバッファに入れ，（７２１１：７２１２）が示すバッファ内のディレクトリスロットに次のものを設定します．

●ファイル名（０２ＤＥ～０２Ｅ５）
●ファイルタイプ（０２ＤＢ）とアスキーフラグ（０２ＤＣ）
●＄７９９０を呼び出し，その復帰情報で返されるクラスタ番号
●ファイルモードがランダム〔(０２ＤＡ)＝＄４０〕ならランダムアクセスフラグ（＝＄ＦＦ）

以上のものを設定した後，バッファの内容を再びセクタへ書き込みます．＄７８Ａ９からは，＄７９９０で変更したＤＲＶＴＢＬの内容を＄７５ＦＢによってＦＡＴに記入しています．＄７８Ａ９から呼び出す場合は，ＣＣを除く全てのレジスタが保存されます．なお，このルーチンを呼び出す際は（７１ＦＥ）にドライブ番号，(７２０２)に＄０１(トラック１)を設定しておく必要があります．
※ディレクトリスロット内バイト位置に関しては，文法書２．１０．３を御参照下さい．

（１０） 空きクラスタ検索・設定ルーチン

アドレス ＄７９９０～＄７９Ａ９
レジスタ Ａ，Ｂ，Ｘ

解　説　（７１ＦＥ）の示すドライブのＤＲＶＴＢＬを検索し，未使用クラスタ（＝＄ＦＦ）を発見した場合は，そこに＄Ｃ０を書き込みます（＄７８５Ｆではこのルーチンを呼び出した後，ランダムモードの場合は，そこに＄ＦＤを書き込みます）．このルーチンは復帰情報として，Ａレジスタにクラスタ番号と，Ｘレジスタに上記ＤＲＶＴＢＬ内の該当クラスタを指すアドレスが返されます．

（１１） クラスタ番号，クラスタ内セクタ番号→トラック番号，セクタ番号変換ルーチン

アドレス ＄７９ＡＡ～＄７９Ｃ１
レジスタ Ａ，Ｂ，Ｘ

解　説　（ＦＢＵＦ＋３）のクラスタ番号と（ＦＢＵＦ＋４）のセクタ番号から，トラック番号を（７２０２），セクタ番号－１を（７２００）に返します．入力情報として，ＸレジスタにＦＢＵＦの先頭アドレスを入れておきます．

4—5—6　サンプル（ディスク エディタ）

ディスクをセクタ単位で，編集するプログラムです．データバッファとして＄４０００～＄４１００の領域を使用しています．なお，カーソル移動キーによって任意の場所にカーソルを移動することができます．実行例は，４－３－５のサンプルの実行例とまとめてあります．

```
10 'DISK EDITOR
20 '
30 WIDTH 80,20 'This command is necessary in this progaram
40 '*** Input & Sector Read ***
50   GOSUB 420 ' key input
60   EXEC &H5000
70 '*** Screen Format ***
80   PRINT "DISK EDITOR FOR FM-7"
90   PRINT TAB(10);:PRINTUSING "DRIVE :##  TRACK :##   SECTOR :##";DRV;TRK;SCT
100   PRINT TAB(4);
110   FOR I= 0 TO 15 :PRINT " +";HEX$(I);:NEXT I:PRINT
120   FOR I= &H00 TO &HF0 STEP &H10
130     PRINT RIGHT$("0"+HEX$(I),2);" : ";
140     FOR J= &H00 TO &H0F
150       PRINT RIGHT$("0"+HEX$(PEEK(&H4000+I+J)),2);" ";
160     NEXT J
170     PRINT " :";
180     FOR J= &H00 TO &H0F
190       A= PEEK(&H4000+I+J)
200       IF A<&H20 OR A=&H7F OR A=&HFF THEN A=&H2E
210       PRINT CHR$(A);
220     NEXT J
230     PRINT
240   NEXT I
250   PRINT "KEY 'Q' - ESCAPE THIS MODE  'L' - HARDCOPY ";
260 '*** Edit ***
270   EXEC &H5018
280 '*** Sector Write ***
290   LOCATE 0,19: INPUT "DO YOU WRITE THIS TO DISK (Y/N)";A$
300   IF A$<>"Y" AND A$<>"y" THEN 340
310   INPUT "DO YOU WRITE SAME DISK (Y/N)";A$
320   IF A$="N" OR A$="n" THEN GOSUB 420
330   EXEC &H500F
340 '*** Select Continue ***
350   INPUT "DO YOU CONTINUE (Y/N)";A$
360   IF A$<>"N" AND A$<>"n" THEN 40
370   PRINT "BYE!":END
380 '*** Error ***
390   COLOR 2: PRINT"ERROR OCCURED IN "+A$+" NUMBER !":BEEP:COLOR 7
400   GOTO 420
410 '*** Procedure Key Input ***
420   INPUT "DRIVE :",DRV
430   INPUT "TRACK :",TRK
440   INPUT "SECTOR:",SCT
450   IF DRV>PEEK(&H71FA) OR DRV <0 THEN A$="DRIVE":GOTO 390
460     POKE &H71FE,DRV
470   IF TRK> 39 OR TRK < 0 THEN A$="TRACK":GOTO 390
480     POKE &H7202,TRK
490   IF SCT> 32 OR SCT < 1 THEN A$="SECTOR":GOTO 390
500     POKE &H7200,SCT-1
510   RETURN
```

```
PAGE  001 (821201,000657) DSKEDT

00010                               NAM    DSKEDT  Ver1.0
00020                               OPT    MEM
00030  5000                         ORG    $5000
00040                        *
00050                        * SECTOR READ
00060  5000 86   02                 LDA    #$02     REQEST = DREAD
00070  5002 B7   71FF               STA    $71FF
00080  5005 8E   4000               LDX    #$4000   DATABUFFER = $4000
00090  5008 BF   71FC               STX    $71FC
00100  500B BD   74CE               JSR    $74CE    DISK I/O
00110  500E 39                      RTS
00120                        *
00130                        * SECTOR WRITE
00140  500F 86   03                 LDA    #$03     REQEST = DWRITE
00150  5011 B7   71FF               STA    $71FF
00160  5014 BD   74CE               JSR    $74CE    DISK I/O
00170  5017 39                      RTS
00180                        *
00190                        * SCREEN EDIT
00200  5018 CC   0503               LDD    #$0503   CURSOR X=6,Y=3
00210  501B BD   DA6D               JSR    $DA6D    CURSOR SET
00220  501E F6   05A8               LDB    $5A8
00230  5021 CA   01                 ORB    #$01     CONSOLE FLAG SET
00240  5023 F7   05A8               STB    $5A8
00250  5026 BD   DAF9               JSR    $DAF9    CURSOR ON
00260            5029        REPEAT EQU    *
00270  5029 BD   DA70               JSR    $DA70    CURSOR SET
00280  502C 8E   0500               LDX    #$0500
00290            502F        WAIT   EQU    *        WAIT FOR CURSOR BLINK
00300  502F 30   1F                 LEAX   -1,X
00310  5031 26   FC   502F          BNE    WAIT
00320  5033 BD   DB54               JSR    $DB54    KEY IN
00330  5036 81   1C                 CMPA   #$1C
00340  5038 1027 008C 50C8          LBEQ   RIGHT    IF CURSOR KEY THEN MOVE
00350  503C 81   1D                 CMPA   #$1D
00360  503E 1027 00AD 50EF          LBEQ   LEFT
00370  5042 81   1E                 CMPA   #$1E
00380  5044 27   63   50A9          BEQ    UP
00390  5046 81   1F                 CMPA   #$1F
00400  5048 27   6E   50B8          BEQ    DOWN
00410  504A 81   4C                 CMPA   #$4C
00420  504C 27   22   5070          BEQ    HARDC    "L" THEN HARDCOPY
00430  504E 81   51                 CMPA   #$51
00440  5050 27   1D   506F          BEQ    RETURN   "Q" THEN QUIT EDITOR MODE
00450                        *
00460  5052 81   30                 CMPA   #$30     CHECK KEY "0" < INKEY$ < "F"
00470  5054 25   D3   5029          BLO    REPEAT   & CHANGE INKEY$ TO FIGURE
00480  5056 81   39                 CMPA   #$39
00490  5058 23   0A   5064          BLS    P1
00500  505A 81   41                 CMPA   #$41
00510  505C 25   CB   5029          BLO    REPEAT
00520  505E 81   46                 CMPA   #$46
00530  5060 22   C7   5029          BHI    REPEAT
00540  5062 20   04   5068          BRA    P2
00550            5064        P1     EQU    *
00560  5064 80   30                 SUBA   #$30
00570  5066 20   02   506A          BRA    P3
00580            5068        P2     EQU    *

00590  5068 80   37                 SUBA   #$37
00600            506A        P3     EQU    *        INPUT DATA TO BUFFER & OUTPUT
00610  506A 8D   0A   5076          BSR    INPUT
00620  506C 16   00A4 5113          LBRA   OUTPUT
00640                        *
00650            506F        RETURN EQU    *        RETRUN TO BASIC
00660  506F 39                      RTS
00670                        *
00680            5070        HARDC  EQU    *        HARDCOPY TO PRINTER
00690  5070 4F                      CLRA            CHRACTER ONLY
00700  5071 BD   ADDC               JSR    $ADDC    HARDC
00710  5074 20   B3   5029          BRA    REPEAT
00720                        *
00730            5076        INPUT  EQU    *        INPUT DATA TO BUFFER
```

```
00740  5076 34   02                PSHS   A
00750  5078 8E   4000              LDX    #$4000   X REGISTER IS BUFFER START ADD
00760  507B 17   00D4 5152         LBSR   TABLE
00770  507E 3A                     ABX             CALCULATE CHANGING POINT TO X
00780  507F 34   02                PSHS   A
00790  5081 17   00C6 514A         LBSR   LOCATE
00800  5084 86   10                LDA    #$10
00810  5086 3D                     MUL
00820  5087 3A                     ABX
00830  5088 35   02                PULS   A
00840  508A 4D                     TSTA            CHANGE CHANGING POINT
00850  508B 27   0A   5097         BEQ    UPPER
00860  508D A6   84                LDA    ,X       CHANGE LOWER OF BYTE
00870  508F 84   F0                ANDA   #$F0
00880  5091 AB   E4                ADDA   ,S
00890  5093 A7   84                STA    ,X
00900  5095 35   82                PULS   PC,A
00910            5097       UPPER  EQU    *
00920  5097 A6   84                LDA    ,X       CHANGE UPPER OF BYTE
00930  5099 84   0F                ANDA   #$0F
00940  509B E6   E4                LDB    ,S
00950  509D 58                     ASLB
00960  509E 58                     ASLB
00970  509F 58                     ASLB
00980  50A0 58                     ASLB
00990  50A1 34   04                PSHS   B
01000  50A3 AB   E0                ADDA   ,S+
01010  50A5 A7   84                STA    ,X
01020  50A7 35   82                PULS   PC,A
01030                       *
01040            50A9       UP     EQU    *        UP CURSOR
01050  50A9 17   009E 514A         LBSR   LOCATE
01060  50AC 5D                     TSTB
01070  50AD 27   01   50B0         BEQ    ESCUP
01080  50AF 5A                     DECB
01090            50B0       ESCUP  EQU    *
01100  50B0 CB   03                ADDB   #$03
01110  50B2 F7   030C              STB    $30C
01120  50B5 16   FF71 5029         LBRA   REPEAT
01130                       *
01140            50B8       DOWN   EQU    *        DOWN CURSOR
01150  50B8 17   00BF 514A         LBSR   LOCATE
01160  50BB C1   0F                CMPB   #$0F
01170  50BD 27   01   50C0         BEQ    ESCDWN

01180  50BF 5C                     INCB
01190            50C0       ESCDWN EQU    *
01200  50C0 CB   03                ADDB   #$03
01210  50C2 F7   030C              STB    $30C
01220  50C5 16   FF61 5029         LBRA   REPEAT
01230                       *
01240            50C8       RIGHT  EQU    *        RIGHT CURSOR
01250  50C8 17   0087 5152         LBSR   TABLE
01260  50CB 4D                     TSTA
01270  50CC 27   0E   50DC         BEQ    RIGHT1
01280  50CE C1   0F                CMPB   #$0F
01290  50D0 27   10   50E2         BEQ    RGTDWN
01300  50D2 B6   030B              LDA    $30B
01310  50D5 8B   02                ADDA   #$02
01320  50D7 B7   030B              STA    $30B
01330  50DA 20   03   50DF         BRA    ESCRGT
01340            50DC       RIGHT1 EQU    *
01350  50DC 7C   030B              INC    $30B
01360            50DF       ESCRGT EQU    *
01370  50DF 16   FF47 5029         LBRA   REPEAT
01380            50E2       RGTDWN EQU    *        IF RIGHT END THEN DOWN LINE
01390  50E2 8D   66   514A         BSR    LOCATE
01400  50E4 C1   0F                CMPB   #$0F
01410  50E6 27   F7   50DF         BEQ    ESCRGT
01420  50E8 86   05                LDA    #$05
01430  50EA B7   030B              STA    $30B
01440  50ED 20   C9   50B8         BRA    DOWN
01450                       *
01460            50EF       LEFT   EQU    *        LEFT CURSOR
01470  50EF 8D   61   5152         BSR    TABLE
```

```
01480  50F1 4D                      TSTA
01490  50F2 26   0D    5101         BNE    LEFT1
01500  50F4 5D                      TSTB
01510  50F5 27   10    5107         BEQ    LEFUP
01520  50F7 B6   030B               LDA    $30B
01530  50FA 80   02                 SUBA   #$02
01540  50FC B7   030B               STA    $30B
01550  50FF 20   03    5104         BRA    ESCLEF
01560            5101        LEFT1  EQU    *
01570  5101 7A   030B               DEC    $30B
01580            5104        ESCLEF EQU    *
01590  5104 16   FF22 5029          LBRA   REPEAT
01600            5107        LEFUP  EQU    *        IF LEFT END THEN UP LINE
01610  5107 8D   41    514A         BSR    LOCATE
01620  5109 5D                      TSTB
01630  510A 27   F8    5104         BEQ    ESCLEF
01640  510C 86   33                 LDA    #$33
01650  510E B7   030B               STA    $30B
01660  5111 20   96    50A9         BRA    UP
01670                        *
01680            5113        OUTPUT EQU    *        OUTPUT TO SCRN
01690  5113 81   09                 CMPA   #$09
01700  5115 22   04    511B         BHI    OUTPT1
01710  5117 8B   30                 ADDA   #$30
01720  5119 20   02    511D         BRA    OUTPT2
01730            511B        OUTPT1 EQU    *
01740  511B 8B   37                 ADDA   #$37
01750            511D        OUTPT2 EQU    *

01760  511D BD   D9D9               JSR    $D9D9
01770                        *
01780  5120 B6   030B               LDA    $30B     OUTPUT CHARACTER OF THE POINT
01790  5123 34   02                 PSHS   A
01800  5125 8D   2B    5152         BSR    TABLE
01810  5127 CB   37                 ADDB   #$37
01820  5129 F7   030B               STB    $30B
01830  512C BD   DA70               JSR    $DA70
01840  512F A6   84                 LDA    ,X
01850  5131 81   FF                 CMPA   #$FF
01860  5133 27   08    513D         BEQ    OUTPT3
01870  5135 81   7F                 CMPA   #$7F
01880  5137 27   04    513D         BEQ    OUTPT3
01890  5139 81   20                 CMPA   #$20
01900  513B 24   02    513F         BHS    OUTPT4
01910            513D        OUTPT3 EQU    *
01920  513D 86   2E                 LDA    #$2E
01930            513F        OUTPT4 EQU    *
01940  513F BD   D9D9               JSR    $D9D9
01950  5142 35   02                 PULS   A
01960  5144 B7   030B               STA    $30B
01970  5147 16   FF7E 50C8          LBRA   RIGHT
01980                        *
01990            514A        LOCATE EQU    *        GET LOCATE OF CUSOR
02000  514A FC   030B               LDD    $30B     A <= X (-5)   B <= Y (-3)
02010  514D 80   05                 SUBA   #$05
02020  514F C0   03                 SUBB   #$03
02030  5151 39                      RTS
02040                        *
02050            5152        TABLE  EQU    *        GET POINT OF TABLE
02060  5152 8D   F6    514A         BSR    LOCATE   A <= UPPER (=0) / LOWER (=1)
02070  5154 5F                      CLRB
02080            5155        RPTAB  EQU    *
02090  5155 81   03                 CMPA   #$03
02100  5157 25   05    515E         BLO    ESTAB
02110  5159 80   03                 SUBA   #$03
02120  515B 5C                      INCB
02130  515C 20   F7    5155         BRA    RPTAB
02140            515E        ESTAB  EQU    *
02150  515E 39                      RTS
02160                        *
02170                               END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=515E
PROGRAM ENTRY ADDR=****
```

4—6　コンソール入出力・プリンタ出力ルーチン

4—6—1　ＢＩＯＳ関連ルーチン

コンソールの入出力では，ＢＩＯＳに対するＲＣＢやサブシステムへのコマンドおよびパラメータを設定するルーチンが，その具体的な実行ルーチンとなっています．従って，サブシステムに対して機能する**ＳＵＢＯＵＴ**，**ＳＵＢＩＮ**が頻繁に用いられています．これらを呼び出すためのＲＣＢは（０５Ａ０～０５Ａ７）に設定されます（ＲＯＭ内に持っていることもあります）．また，ＳＵＢＯＵＴのときのデータバッファは（**０１００～０１７Ｆ**）に，ＳＵＢＩＮのときのデータバッファは（**０１Ｂ５～０１Ｂ９**）に設定されます．

（１）　汎用ＲＣＢ設定ルーチン

- アドレス　**＄Ｄ９Ｆ４～＄ＤＡ２０**
- 機　能　**入出力のＲＣＢをワークエリア（０５Ａ０～０５Ａ７）に設定する．**
- レジスタ　**Ａ，Ｂ，Ｘ，Ｕ**
- 入力情報　**Ｕレジスタ＝ＲＣＢの書かれている先頭アドレス**
- ＷＯＲＫ　**（０５Ａ０～０５Ａ７）＝汎用ＲＣＢ設置エリア　Ⓦ**

- 解　説　Ｕレジスタの値により，下表のようなＲＣＢが設定されます．

表４・６・１　設定されるＲＣＢの内容

Ｕレジスタ	ＲＱＮＯ	ＲＣＢＳＴＡ	ＲＣＢＤＢＡ	ＲＣＢＬＮＨ	ＲＣＢＢＭＨ
	５Ａ０	５Ａ１	５Ａ２ ５Ａ３	５Ａ４ ５Ａ５	５Ａ６ ５Ａ７
＄ＤＡ０１	＄１２　ＩＮＰＵＴ	００	＄０４ ３Ｄ	＄００ ００	＄０１ ０４
＄ＤＡ０９	＄１３　ＩＮＰＵＴＣ	００	＄０４ ３Ｄ	＄００ ００	＄０１ ０４
＄ＤＡ１１	＄１１　ＳＵＢＩＮ	００	＄０１ Ｂ５	＄００ ０３	＄００ ０５
＄ＤＡ１９	＄１０　ＳＵＢＯＵＴ	００	＄０１ ００	＄００ ０４	＄００ ８０

(2)　SUBOUT　PUT設定ルーチン

アドレス	$DA21～$DA2F
機　能	BIOSに対してはSUBOUTを要求するRCBを，サブシステムに対してはPUT要求〔システム仕様2．2．11(3)〕のRCBを設定する．
レジスタ	A，B，X，U（全て保存）
WORK	（0101：0102）Ⓦ
SUB	$D9F4→汎用RCB設定ルーチン

解　説　Uレジスタに$DA19を入れて汎用RCB設定ルーチンを呼び，SUBOUTのRCBを設定した後，（0101：0102）に$0003を入れ，データの継続のないPUT要求の設定をします．

(3)　SUBOUT　PUT初期設定ルーチン

アドレス	$DA30～$DA35
機　能	$DA21の機能に加えてデータバイト数を0に初期設定する．
レジスタ	A，B，X，U（全て保存）
WORK	（0103）Ⓦ
SUB	$DA21→SUBOUT　PUT設定ルーチン

(4)　SUBOUT　PUT後続設定ルーチン

アドレス	$DA36～$DA61
機　能	PUTの出力文字列を設定していく．
レジスタ	A，B，X（全て保存）
入力情報	Aレジスタ＝出力文字コード，またはオーダコード
WORK	（0101：0102）Ⓦ　，（0103）Ⓡ／Ⓦ （0104～　）Ⓦ　，（05A5）Ⓦ
SUB	$DA64→BIOS実行ルーチン

解　説　（0104～）PUTの文字列を蓄えていきますが，文字列が124バイトを越えたときは，**継続フラグ**を立てて出力し（$DA64），CONT

ＩＮＵＥコマンドの設定をした後に，再び（０１０４）以降に蓄えていきます．
文字列を蓄えていく際に，データバイト数（０１０３），およびＲＣＢ中のデータバイト数（０５Ａ５）を＋１していきます．

（５）　ＢＩＯＳ実行ルーチン

アドレス	＄ＤＡ６２，＄ＤＡ６４〜＄ＤＡ６Ｃ
機　能	設定されたＲＣＢ，コマンドをＢＩＯＳに実行させる．
レジスタ	Ａ，Ｘ（Ｘは保存）
ＳＵＢ	＄ＤＡ３６→ＳＵＢＯＵＴ　ＰＵＴ後続設定ルーチン ＄００ＤＥ→ＢＩＯＳジャンプルーチン

解　説　＄ＤＡ６２がエントリの場合は，＄ＤＡ３６を呼び出しているので，Ａレジスタに文字コード等を入れておく必要があります．また，＄ＤＡ６４からは，（０５Ａ０〜０５Ａ７）のＲＣＢの内容をＢＩＯＳに実行させます（このときＡレジスタは使用しません）．
（３）〜（５）のルーチンの具体的な使用法は，まず＄ＤＡ３０を呼んでＲＣＢなどを設定した後，Ａレジスタに文字コードなどを入れて＄ＤＡ３６を呼び出すという動作を繰り返し，最後に＄ＤＡ６４，あるいは＄ＤＡ６２を呼んでＢＩＯＳに実行させるという形が一般的なようです．

４—６—２　コンソール１バイト入出力ルーチン

（１）　スクリーン１バイト出力ルーチン

アドレス	＄Ｄ９Ｄ９〜＄Ｄ９ＤＤ
機　能	サブシステムのＰＵＴを用いてスクリーンに１バイト出力する．
レジスタ	Ａ
入力情報	Ａレジスタ＝出力する文字コード
ＳＵＢ	＄ＤＡ３０→ＳＵＢＯＵＴ　ＰＵＴ初期設定ルーチン ＄ＤＡ６２→ＢＩＯＳ実行ルーチン

(2)　キーボード1バイト入力ルーチン

アドレス	＄DB54～＄DB58
機　　能	キー入力があるまで待ち，キー入力があったときは，キーボードから1バイト入力する．※キー入力待ちルーチンとしても使用可能．
レジスタ	A，B，X
復帰情報	Aレジスタ＝入力キーコード
SUB	＄DB6D→キー入力ルーチン

(3)　キー入力ルーチン

アドレス	＄DB6D～＄DB83
機　　能	キー入力の有無にかかわらず，キーボードから1バイト入力する（BIOSのKEYIN〔システム仕様2．1．4（7)〕を使用)．
レジスタ	A，B，X
復帰情報	Aレジスタ＝入力キーコード（キー入力があったとき） Zフラグ　＝キー入力がない場合にセットされる
WORK	(05A0～05A5)＝BIOSへのRCB　Ⓦ (01B5：01B6)＝データバッファ　Ⓡ
SUB	＄00DE→BIOSジャンプルーチン

(4)　フィールド設定ルーチン

アドレス	＄D92F～＄D93C
機　　能	フィールドの始まりを設定する．
レジスタ	A
WORK	(01E5)＝フォアグラウンドカラー　Ⓡ
SUB	＄DA30→SUBOUT　PUT初期設定ルーチン ＄DA36→SUBOUT　PUT後続設定ルーチン ＄DA62→BIOS実行ルーチン

解　　説　サブシステムにオーダとしてSF（Set field）を出力し，**アトリビ**

ュート文字として，（０１Ｅ５）の内容を出力します．フィールドの設定をすることには重要な意味があり，連続した文字列でも，フィールドが異なっていると，サブシステムは，２つの別の文字列として処理します．従って，新規に出力する文字列を新たなフィールドとして定義したい場合は，このルーチンを呼ぶ必要があります〔システム仕様２．２．４（５）～（９）参照〕．

（５） フィールド設定，およびイレーズルーチン

アドレス	＄Ｄ９３Ｄ～＄Ｄ９４３
機　能	フィールドを設定した後，ＥＬ〔Erase Line；システム仕様２．２．４（１２）〕を出力し，フィールドの終りまでイレーズを行う．
レジスタ	Ａ
ＳＵＢ	＄Ｄ９２Ｆ→フィールド設定ルーチン ＄Ｄ０８Ｅ→１バイト出力ルーチン

４―６―３　その他の主なコンソール入出力関係ルーチンの概要

（１） カーソルのＸ，Ｙ座標読み取りルーチン

アドレス	＄Ｄ９ＤＥ～＄Ｄ９Ｆ３
機　能	現在のカーソル位置（バッファアドレス）を読み取る．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ　（Ｘ，Ｕは保存）
復帰情報	Ｄレジスタ＝カーソルのＸ座標とＹ座標
ＷＯＲＫ	（０１Ｂ７）　＝サブシステムへのコマンド　Ⓦ （０１Ｂ８：０１Ｂ９）＝サブコマンドからの復帰情報　Ⓡ （０３０Ｂ：０３０Ｃ）＝カーソルの（Ｘ座標：Ｙ座標）　Ⓦ
ＳＵＢ	＄Ｄ９Ｆ４→汎用ＲＣＢ設定ルーチン ＄００ＤＥ→ＢＩＯＳジャンプルーチン

解　説　＄Ｄ９Ｆ４を使いＳＵＢＩＮのＲＣＢを作った後，ＧＥＴ Buffer Addressのコマンドをサブシステムに送ります〔システム仕様２．２．１１（１０）〕．その復帰情報から，バッファアドレスのＸおよびＹ座標，即ち**カーソルの位置**を読み取ってワークエリア（０３０Ｂ：０３０Ｃ）へ格納します．

（２） カーソルのＸ，Ｙ座標設定ルーチン

アドレス	＄ＤＡ６Ｄ，＄ＤＡ７０～＄ＤＡ７Ｆ
機　　能	カーソルの位置（バッファアドレス）を設定する．
レジスタ	Ａ，Ｂ
入力情報	Ｄレジスタ＝カーソルのＸ座標とＹ座標（＄ＤＡ６Ｄの場合のみ）
ＷＯＲＫ	（０３０Ｂ：０３０Ｃ）＝カーソルの（Ｘ座標：Ｙ座標）　Ⓡ／Ⓦ
ＳＵＢ	＄ＤＡ３０→ＳＵＢＯＵＴ　ＰＵＴ初期設定ルーチン ＄ＤＡ３６→ＳＵＢＯＵＴ　ＰＵＴ後続設定ルーチン ＄ＤＡ６２→ＢＩＯＳ実行ルーチン

解　　説　ＳＢＡ〔Set Buffer Address；システム仕様２．２．４（１０）〕を用いて，（０３０Ｂ：０３０Ｃ）に格納されている，カーソルのＸ座標とＹ座標をサブシステムにバッファアドレスとして送ります．＄ＤＡ６Ｄがエントリの場合にはＤレジスタの値を事前に（０３０Ｂ：０３０Ｃ）に代入しています．

（３） 画面消去（ＣＬＳ）ルーチン

アドレス	＄ＤＡ８０，＄ＤＡ８Ｂ～＄ＤＡＡ５
機　　能	画面およびコンソールバッファを初期化する．
レジスタ	Ａ，Ｂ
入力情報	Ｂレジスタ＝消去範囲（解説参照；＄ＤＡ８Ｂエントリの場合）
ＷＯＲＫ	（０１０２～０１０５）＝サブシステムへのデータバッファ　Ⓦ （０１Ｅ５：０１Ｅ６）＝(フォア：バック)グラウンドカラーⓇ （０５Ａ５）＝ＲＣＢ中のデータバイト数　Ⓦ
ＳＵＢ	＄００Ｄ８→テキスト読み込みルーチン ＄Ｃ７ＢＥ→テキスト数値読み込みルーチン ＄ＤＡ２１→ＳＵＢＯＵＴ　ＰＵＴ設定ルーチン ＄ＤＡ６４→ＢＩＯＳ実行ルーチン

解　　説　＄ＤＡ８０は「ＣＬＳ」のエントリですので，テキストの読み込みをしています．ユーザは＄ＤＡ８Ｂをエントリとして使うのが適当です．＄ＤＡ２１を呼んではいますが，コマンドをＰＵＴに代えてＥＲＡＳＥ２を再設定しています．入力情報のＢレジスタの値は，ＥＲＡＳＥ２のＷの値と同様です．ま

た，（０１Ｅ５：０１Ｅ６）より画面の文字色と背景色の設定も行います〔システム仕様２．２．１１（１３）参照〕．

（４）　カーソル消去・表示ルーチン１

アドレス	＄ＤＡＤＦ，＄ＤＡＥ４～＄ＤＡＥＥ
機　　能	カーソルを表示（＄ＤＡＥ４），消去（＄ＤＡＤＦ）する．
レジスタ	Ａ，Ｂ　（全て保存）
ＳＵＢ	＄ＤＢＤ５→コンソール制御ルーチン

（５）　カーソル消去・表示ルーチン２

アドレス	＄ＤＡＥＦ，＄ＤＡＦ６～＄ＤＢ０６
機　　能	カーソルを表示（＄ＤＡＦ６），消去（＄ＤＡＥＦ）する．
レジスタ	Ａ，Ｂ，Ｘ　（Ｘは保存）
ＷＯＲＫ	（０５Ａ８）　　　　　＝コンソール制御フラグ　Ⓡ
	（０１Ｂ７：０１Ｂ８）＝データバッファ　　　Ⓦ
ＳＵＢ	＄００ＤＥ→ＢＩＯＳジャンプルーチン

解　　説　（４）のルーチンでは，結果的に（０５Ａ８）を変えてしまいますが，このルーチンは，（０５Ａ８）を参照するだけの一時的なものとなっています．ただし,（０５Ａ８）がもともとカーソル表示をしないように設定されていれば，＄ＤＡＦ６のエントリでこのルーチンを呼んでもカーソルは表示しません．

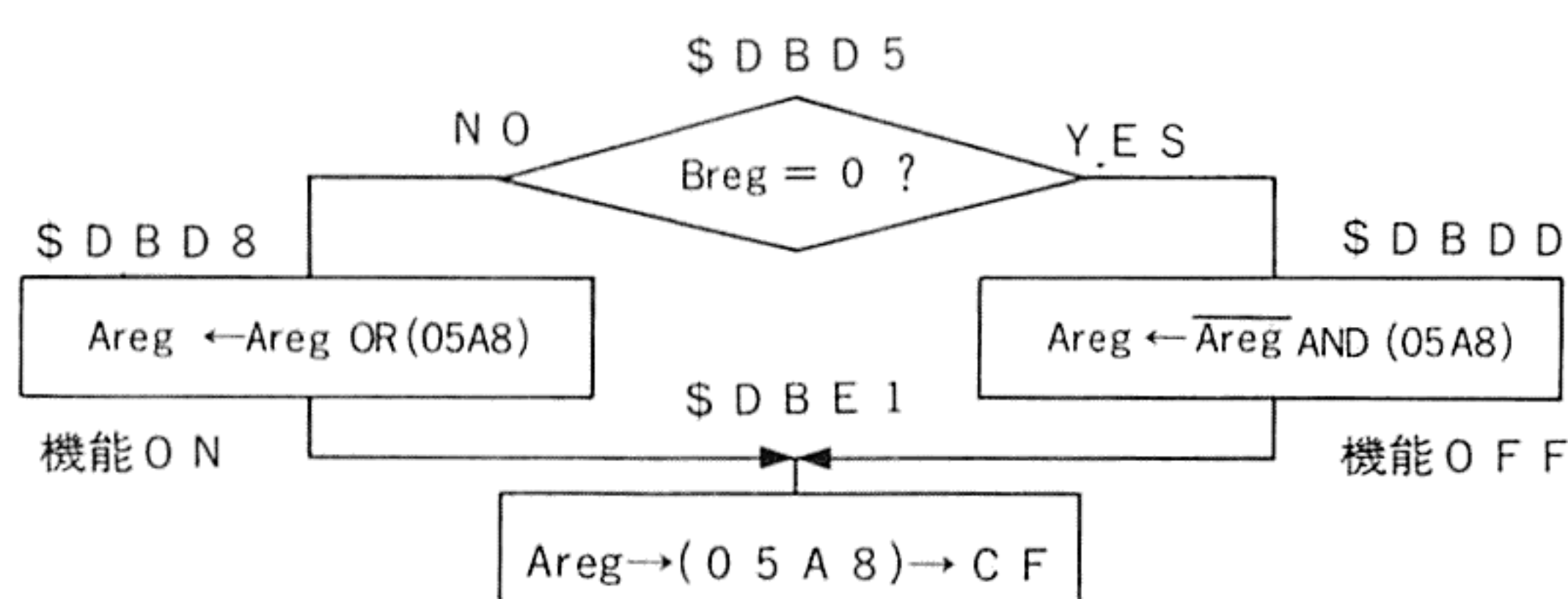
（０５Ａ８）の内容によりコンソール機能を設定するという点から見れば，次の（６）のルーチンと同じ機能を有しているといえます．なお，このルーチンで使うＲＣＢは，＄ＤＢ０７～＄ＤＢ０Ｃに格納されています．

(6) コンソール制御ルーチン

アドレス	$DBD5, $DBD8, $DBDD, $DBE1〜$DBF2
機　能	コンソール機能を選択する.
レジスタ	A, B
入力情報	解説参照
WORK	(05A8)　＝コンソール制御フラグ　Ⓦ
	(0102：0103)＝サブシステムへのデータバッファ　Ⓦ
SUB	$DA21→汎用RCB設定ルーチン
終了条件	画面消去ルーチンの途中($DAA0)にジャンプし, RCB中のデータバイト数の設定を行い, BIOS実行ルーチンを呼び出すという部分を流用している.

解　説　$DBD5エントリの場合は, Bレジスタの内容を見て, その値が0のときは$DBDDに飛び, 0以外のときは$DBD8に飛びます. $DBD8および$DBDDでは, Aレジスタの各ビットで指定される機能のON, OFFを行います. $DBE1はAレジスタの内容が直接に(05A8)とCF(制御フラグ)に代入されて, 各機能が選択されます(図4・6・1). このルーチンは$DA21を呼んではいますが, コマンドをPUTに代えてCONSOLE CONTROL〔システム仕様2. 2. 11 (12)〕で置き換えています.

図4・6・1　各エントリと動作の関係

$DBD5
Breg＝0 ?
NO
YES
$DBD8
Areg ←Areg OR (05A8)
機能ON
$DBDD
Areg ←$\overline{\text{Areg}}$ AND (05A8)
機能OFF
$DBE1
Areg→(05A8)→CF

CF(制御フラグ)
ビット0：カーソル表示
ビット1：オーダ動作
ビット2：TAB動作
ビット3：ページウェイト
ビット4：プットウェイト
ビット5：オートLF
※1＝ON, 0＝OFF

(7)　BEEPルーチン

解　説　$DB2A～$DB53
レジスタ　A，X

解　説　$DB2Aは「BEEP～」用なので，次のエントリを用いて下さい。

- $DB38　一定時間ブザーを鳴らします（サブシステムのBEL（Bell）オーダ〔システム仕様2．2．4（13）を使用〕．
- $DB3F　ブザーをONします〔BIOSのBEEP　ONを使用〕．
- $DB44　ブザーをOFFします〔BIOSのBEEP　OFF使用〕．
- $DB49　一定時間ブザーを鳴らします〔$DB3F，$DB44使用〕．

このルーチンのRCBは，$DB26～$DB29の領域にあります（BEEP　ON，OFFはシステム仕様2．1．4(8)および(9)参照〕．

4—6—4　プリンタ出力ルーチン

(1)　プリンタ　オープン（OPEN）ルーチン

アドレス　$D9A5～$D9D8
機　能　プリンタをオープンする．
レジスタ　A，B，X，Y
入力情報　Aレジスタ＝ファイルのモード〔アウトプット（＝$20）のみ〕
WORK　(02E7：02E8)＝ファイルのオプション　Ⓡ
(05AA：05AB)＝プリンタのジャンプテーブル先頭アドレス（＝$0778）　Ⓡ
(05BF)　＝プリンタ自動改行フラグ　Ⓦ
SUB　$D617→プリンタ1バイト出力ルーチン

解　説　プリンタのジャンプテーブルの先頭アドレスから19バイト目と20バイト目，即ち（078B）と（078C）には，各々のプリンタの**改行桁数**と**フィールド改行桁数**（出力するものをコマンドで区切る場合，BASICは1行を14文字ずつのフィールドに分けますが，このとき現在の桁数がフィールド

改行桁数よりも大きくなった場合に改行されます）が入っていますが，オプションの指定により次のように変わります（“W”の指定のときは，縮小文字を指定しておかないと，不適当な所で改行することになります）．

	改行桁数（078B）	フィールド改行桁数（078C）
“W”：	$88（=136）	$70（=112）
“S”：	$50（=80）	$38（=56）…（BASIC起動時）

また，オプションの第2パラメータ（“F”または“N”）があれば，プリンタの自動改行フラグ（05BF）が，“F”のときは0，“N”のときは1が代入され，PRINTルーチンで参照されます．これらについては，システム解説の14．5節を御参照下さい．この後，このルーチンはプリンタ1バイト出力ルーチンを利用して，プリンタのレディチェックを行っています．

（2） プリンタ1バイト出力ルーチン

アドレス	$D617～$D62A
機　能	プリンタに1バイト出力する．
レジスタ	A，X，Y
入力情報	Aレジスタ＝出力するデータ Xレジスタ＝プリンタジャンプテーブルの先頭アドレス
SUB	$D659→プリンタのレディチェックおよび1バイト出力ルーチン

解　説　$D659を使ってプリンタにデータを出力した後，出力データが文字コード（$20～）なら，ジャンプテーブルの18バイト目（078A）に格納しているプリンタの現在の桁数を＋1します．ただし，改行桁数（078B）より大きくなった場合は，0に再設定されます．

※　このルーチンおよび次の(3)でプリンタに出力した文字データは，プリンタのバッファ内に蓄えられ，$0D（CR：キャリッジリターン）または$0A（LF：ラインフィード）がプリンタに出力された時点，あるいはバッファがいっぱいになった時点で印字されます．

(3) プリンタのレディチェック及び1バイト出力ルーチン

アドレス	$D659～$D66F
機　能	プリンタのレディチェックおよび1バイト出力する.
レジスタ	A, X　(Xは保存)
入力情報	Aレジスタ＝出力するデータ
WORK	(0100)＝データバッファとして　Ⓦ
SUB	$D678→Abort ルーチン
	$00DE→BIOSジャンプルーチン

解　説　BIOSのLPCHK〔システム仕様2.1.4(19)〕を使いレディチェックを行い, レディ状態でなければブレーク・キーをチェックしてそれが押されていればAbortします. 押されていなければLPCHKを繰り返します. レディ状態ならAレジスタの内容をLPOUT〔システム仕様2.1.4(10)〕を使って出力します. このルーチンで使うRCBはLPCHK用のものが$D670～$D671, LPOUT用のものが$D672～$D677に格納されています.

(4) プリンタ改行ルーチン

アドレス	$D64A～$D658　($D62B～$D649)
機　能	プリンタのバッファに蓄えられた文字データを印字した後, 改行する.
レジスタ	A, B, X, Y　(Xは保存)
WORK	(05AA:05AB)＝プリンタジャンプテーブル先頭アドレス Ⓡ
	(01E3)　＝プリンタ出力禁止コード(＝$0D) Ⓡ
SUB	$D62B→プリンタ1バイト出力ルーチン(改行用)

解　説　$D62Bを利用してCR($0D)とLF($0A)を出力します. $D62Bでは(01E3)で出力禁止コードを指定しているので, 実際にはCRが出力されません. ここに$0Aに設定したときには, プリンタのディップスイッチが自動改行になっている場合はCRが出力され2行改行されることになりますが, 自動改行でない場合は, LFが出力されないので改行されません.

第５章　数値演算

5－1　マシン語上での整数の表現

６８０９では，符号なし２進数，符号付き２進数，パック化１０進数の３種の記数法の演算ができるようになっています．ここでは，Ｆ－ＢＡＳＩＣで使用されている符号なし２進数と符号付き２進数について述べていきます．

(1)　符号なし２進数での正の整数の表し方（１バイトについて）

１０進数と２進数との対応は表**5・1・1**を御覧下さい．

数式で表すと

$b_7 \times 2^7 + b_6 \times 2^6 + b_5 \times 2^5 + b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0$

となります．

表5・1・1　符号なし２進数の値

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	10進
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
				∫				∫
1	1	1	1	1	1	1	1	255

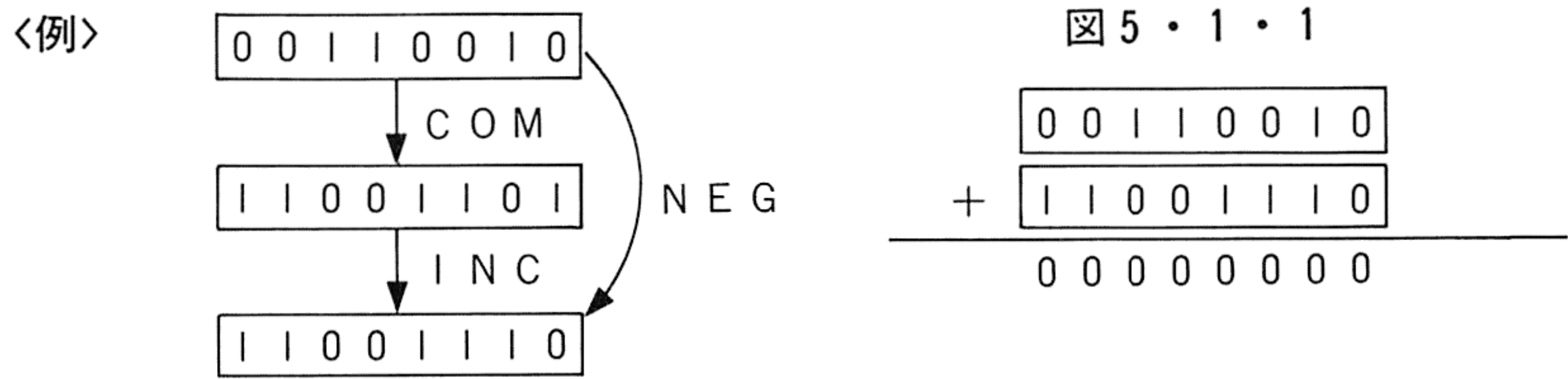
(2)　符号付き２進数での整数の表し方（１バイトについて）

正の整数についてはb_7を０にしてb_6以下で符号なし２進数と同様に表します．負の数については少々複雑ですが例を御覧下さい．

〈例〉

```
00110010 ─┐
   │ COM   │
   ▼       │
11001101   │ NEG
   │ INC   │
   ▼       │
11001110 ◄─┘
```

図5・1・1

```
   00110010
+  11001110
───────────
   00000000
```

例では１０進数で５０という数値がAccに代入してあります．このAccのすべてのビットを反転します(６８０９の命令ではＣＯＭという)．さらにこの数値に１を足します(６８０９の命令ではＩＮＣ)．この結果Accに求められた値を－５０という値にします(この数値の負を求める命令をＮＥＧという)．なぜこれが－５０を表すかは，図**5・1・1**を御覧下さい．２進数で５０という数値とこの－５０という数値を足すと０となります．このことからこの数値を－５０とします．表**5・1・2**は１０進数との対応を示します．

(3)　2バイトの整数

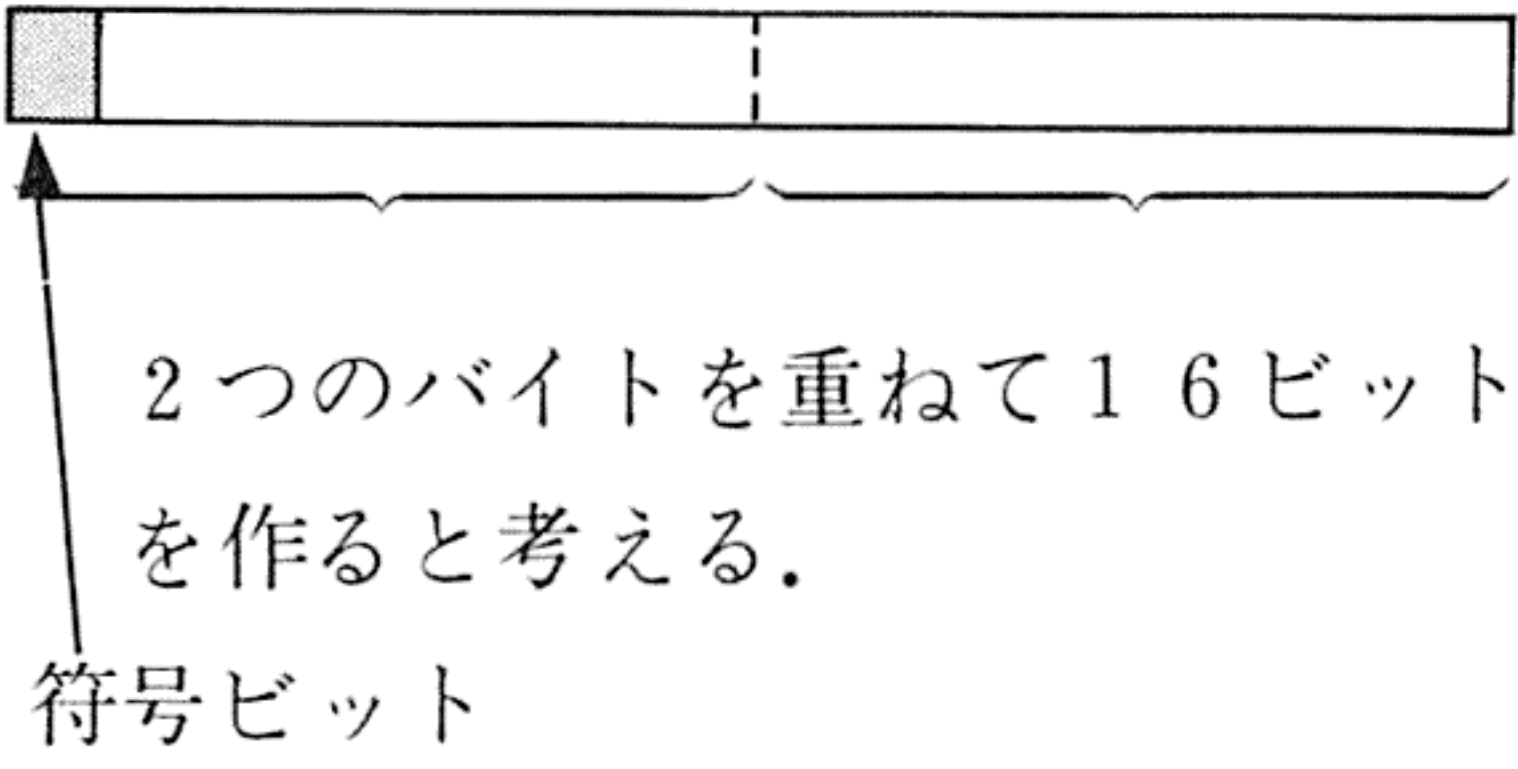
2バイトの整数は符号なし2進数を下位バイトに，符号付き2進数を上位バイトにしたものです．つまり図5・1・2のように16ビットの一連の符号付2進数と考えられます．F－BASICではこのような形で整数を扱っています．

表5・1・2

2進数	10進数
0 1 1 1 1 1 1 1	127
⁝	
0 0 0 0 0 0 0 1	1
0 0 0 0 0 0 0 0	0
1 1 1 1 1 1 1 1	－1
1 1 1 1 1 1 1 0	－2
⁝	
1 0 0 0 0 0 0 0	－128

(4)　整数の減算

6809では減算（SUB）という命令がありますが，この命令を使わなくても減算が可能です．それは，引く方の数値の負をとって（NEGをとる：これからはこの意味でNEGという言葉を使います)，それから加算をすればよいわけです．

図5・1・2

5－2　演算の方法

一般的な演算の方法は次のフローチャートで示されるアルゴリズムを実行しています．

☆数値型の判断

前にも述べたように数値には，整数，単精度実数，倍精度実数の３種類があるためこれを判断しなければなりません．これを調べるには，＄００１７番地を見ます．この値は＄０２のとき整数型，＄０４のとき単精度型，＄０８のとき倍精度型となっています．

☆Error処理

Errorは演算の前後に処理されます．前のErrorはおもにType Mismatchなどで後ろのErrorはOverflowなどです．

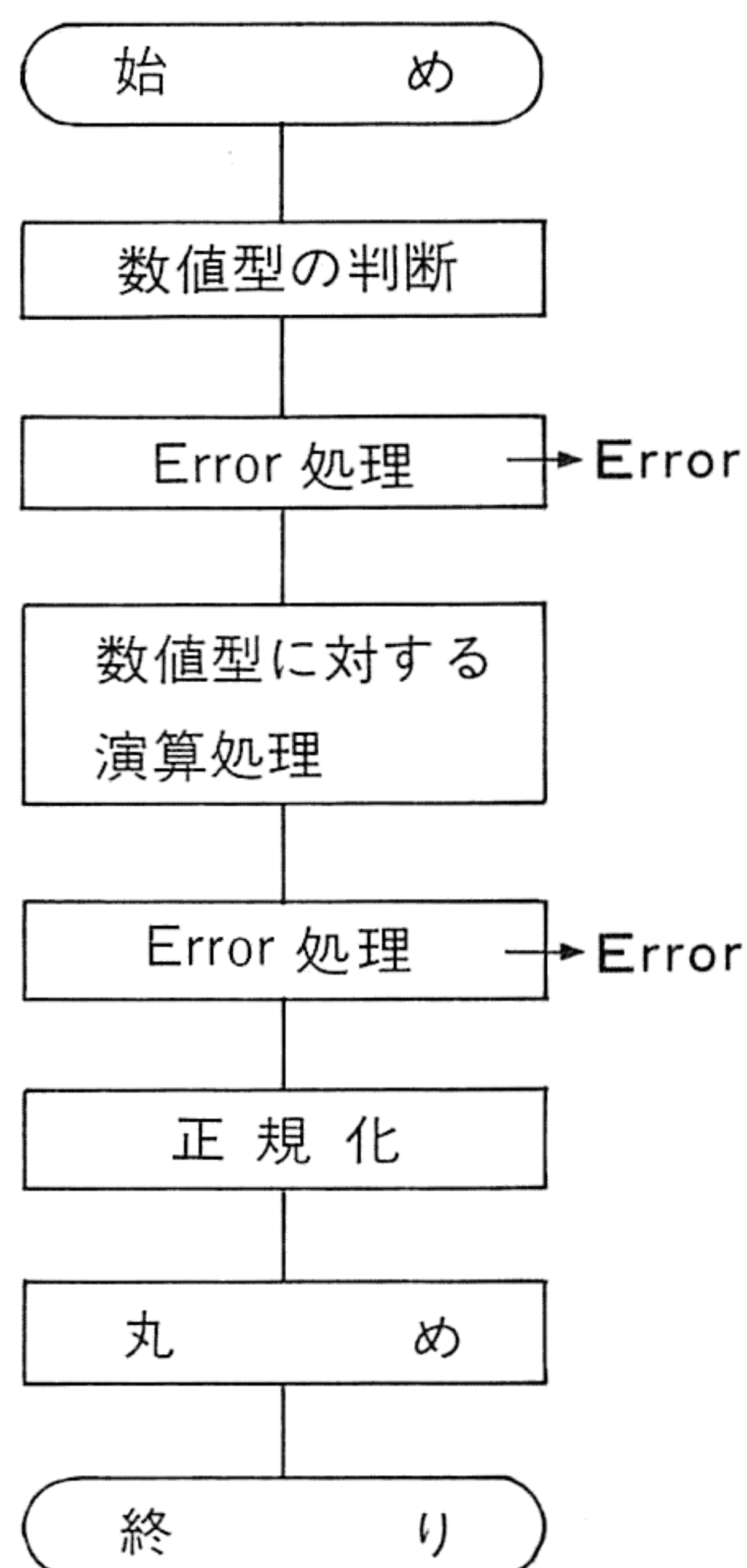
☆正規化ルーチン

後で詳しく述べますのでここでは簡単に触れておきます．このルーチンは一回の演算で起こる誤差をなるべく小さくしようとするルーチンで，演算を終えるたびにこのルーチンを必ず通さなければなりません．

☆丸めルーチン

これも後で詳しく述べますが，四捨五入ルーチンと言うこともできます．

以上が演算を行う上で重要なルーチンですが，ユーザが自分で関数を作るときも（例えばＵＳＲ関数)，このようなアルゴリズムを作ることが必要です（Ｆ－ＢＡＳＩＣ・ＲＯＭ内の関数を使えば問題はありません)．

図５・２・１
演算のフローチャート

5－2－1　数値型判断ルーチン

アドレス	$BCAE, $BCA5, $BCB3
機　　能	$BCAE：文字列型のときにはErrorを発生し，他の型のときには表のようにAレジスタとフラグをセットする． $BCA5：文字列型以外はすべてErrorを発生する． $BCB3：数値の型に従って表のようにフラグとAレジスタの内容が変化する．
レジスタ	A，B，CC
入力情報	(0017)＝FAC1の数値の型（$02＝整数型，$03＝文字列型，$04＝単精度型，$08＝倍精度型）
復帰情報	CC，Aレジスタに表5・2・1の数値が入る．
WORK	(0017)
SUB	$8DD1→エラー処理ルーチンエントリ

表5・2・1　フラグとAレジスタの状態

数値型	$0017	H	N	Z	V	C	Aレジスタ
整数型	$02	0	1	0	0	1	$FF
単精度型	$04	0	0	0	1	1	$01
倍精度型	$08	0	0	0	0	0	$05
文字列型	$03	0	0	1	0	1	$00

解　　説　FAC1の数値型を表す数値はF－BASIC上では$0017番地に格納されます．このルーチンはこの値を調べて各フラグをセットし，必要に応じてError（Type Mismatch）を発生します．

5－2－2　符号判定ルーチン

アドレス	$B3C8
機　　能	FAC1の数値の符号を判定する．
レジスタ	A，B，X
入力情報	FAC1　＝数値 (0017)＝FAC1の数値型

復帰情報　ＣＣとＢレジスタがＦＡＣ１の符号によって下のように復帰する．

表５・２・２

符　号	Ｂレジスタの内容	Ｃフラグ	Ｚフラグ
－	＄ＦＦ	１	０
＋	＄０１	０	０
０	＄００	０	１

WORK　ＦＡＣ１

ＳＵＢ　＄ＢＣＡＥ→数値型判断ルーチン　５－２－１参照

終了条件　文字列型の数値がＦＡＣ１に入っている（＄００１７番地に＄０３が入っている）ときはError（Type Mismatch）が発生する．

解　説　ＦＡＣ１の値は数値であれば何型でも使用可能です．このルーチンはＬＯＧなどのError処理を行ったり，正負によって処理の仕方を変えたいときに用いられます．

５－２－３　正規化ルーチン

アドレス　＄ＡＦ９７，＄Ｂ００Ａ（＄Ｂ００Ａは＄ＡＦ９７とペアで使用）

機　能　＄ＡＦ９７：数値の正規化を行いそのあと丸めを行う．

＄Ｂ００Ａ：数値の丸めを行う．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　＄ＡＦ９７のとき

ＦＡＣ１　　＝正規化されていない数値

（００１５）＝単精度型のとき＄００，倍精度型のとき＄００以外

復帰情報　＄ＡＦ９７のとき

ＦＡＣ１　　＝正規化された数値

WORK　ＦＡＣ１

（００８Ｃ）＝保護バイト

解　説　正規化とか丸めとか聞きなれない名前ですが，数値の誤差を最小限にするためには必要不可欠なものです．

(1) 正規化ルーチン

どんなコンピュータであっても，浮動小数点表示の仮数部には限りがあります．下の例を御覧下さい．

〈例〉

０．１１０１００００×2^{10}＝０．０００１１０１０×2^{13}……①

０．１１０１００１１×2^{10}≒０．０００１１０１０×2^{13}……②

仮数部が８ビット（初めの０と小数点は含まない）であるとします．①では左辺と右辺は同じ数値を表しています．ところが②の左辺のように下位３ビットに数値が現れると，左辺と右辺は等しくなりません．ここで注目してほしいのは右辺の小数点以後の３ビットが無意味になってしまうことです．ですから演算を行った後は，この無意味なビットをなくすため左辺のように必ず小数点直後に１が立つようにします．

Ｆ－ＢＡＳＩＣ上ではＦＡＣ１の他に**保護バイト**と呼ばれるバイトを＄００８Ｃ番地に作っています．このバイトはＦＡＣ１の仮数部の最下位バイトのさらに下位の１バイトを保存しておきます．つまりＦ－ＢＡＳＩＣでは，演算の結果を単精度型では３２ビットまで，倍精度型では６４ビットまで求めています．このようにしてから小数点直後に１が現われるまで全ビット（保護バイトも含む）を左へシフトします（例参照）．

〈例〉　簡単にするため仮数部を８ビット，保護バイトを３ビットとします．

０．０００１１０１０×2^{13}　保護バイト＝０１１

→　０．１１０１００１１×2^{10}……③

このようにすれば②右辺は②左辺と等しくなります．

このような操作を正規化といいます．

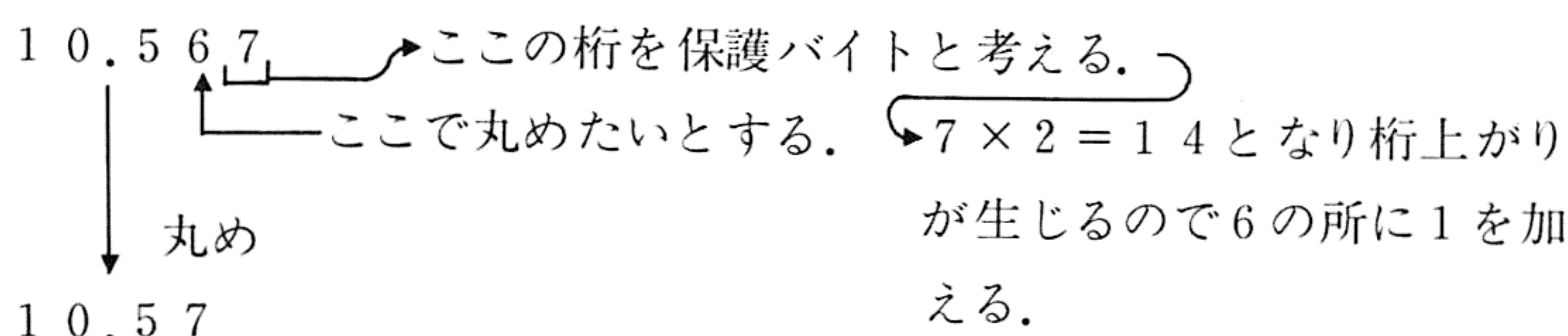
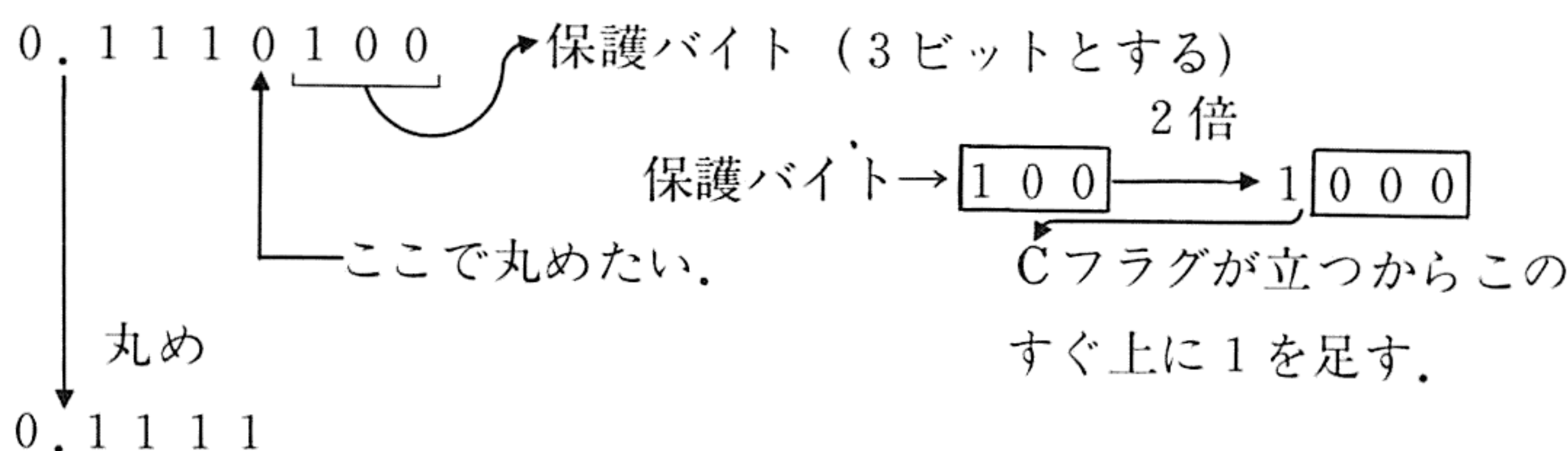
(2) 丸めルーチン

このルーチンは２進数上での四捨五入（２進数では０捨１入か？）のルーチンといってもさしつかえありません．正規化ルーチンでも述べたようにＦＡＣ１には保護バイトがありますが，正規化ルーチンを通した後でもまだ保護バイトに数値が残っていることがあります．このとき保護バイトを２倍してＣフラグが立つかどうかを調べます．Ｃフラグが立ったときにはＦＡＣ１の最下位バイトに１を加えます．この操作を丸めといいます（例参照）．

〈例〉　１０進数の丸め

１０．５６７　→ここの桁を保護バイトと考える．
　　↑ここで丸めたいとする．　→７×２＝１４となり桁上がりが生じるので６の所に１を加える．
丸め
１０．５７

〈例〉　２進数の丸め

０．１１１０１００　→保護バイト（３ビットとする）
　　↑ここで丸めたい．
保護バイト→［１００］―２倍→１［０００］
Ｃフラグが立つからこのすぐ上に１を足す．
丸め
０．１１１１

５－２－４　切捨てルーチン

アドレス	＄Ｂ０６８
機　能	ＦＡＣの絶対値の切り捨てを行う．
レジスタ	Ａ，Ｂ，Ｘ
入力情報	Ｂレジスタ　＝切り捨てたいビット数のＮＥＧの値
	Ｘレジスタ　＝３つあるＦＡＣの先頭番地（＄００７４，＄００８２，＄００２８）
	ＦＡＣ　＝切り捨てたい数値
	（００１５）＝単精度型のとき＄００，倍精度型のとき＄００以外
	（００８１）＝必ず＄００代入
復帰情報	ＦＡＣ　＝絶対値の切り捨てを行った結果
	Ｂレジスタ　＝＄００
ＷＯＲＫ	（００８Ｃ）＝保護バイト

解　説　Ｂレジスタの値は切り捨てたいビットのＮＥＧ（例えば１ビット切り捨てたいのなら＄ＦＦ，２ビットなら＄ＦＥを代入します）を取った値を入れます．

実際にどのようになるか簡単な例で示します．

〈例〉　Bレジスタに＄FDを入れます（簡単にするため仮数を8ビットとします初めの0と小数点は数えません）．

$0.10000011 \times 2^3 \quad \rightarrow \quad 0.00010000 \times 2^5$

ＦＡＣには左辺の数値が入っているとします．このルーチンを呼び出すと右辺の数値がＦＡＣに入ります．この数値がＦＡＣ１の中にあるときはこの後＄００８Ｃ番地に＄００を入れて，正規化ルーチン（５－２－３参照）をコールすれば以下のような切り捨てた値がＦＡＣ１に入ります．

0.10000000×2^3

なお，このルーチンは単精度型と倍精度型の数値でしか利用できません．

５－２－５　数値型変換ルーチン

Ｆ－ＢＡＳＩＣではＣＩＮＴ，ＣＳＮＧ，ＣＤＢＬなど数値の数値型を変える命令がありますが，これらは以下の４つのサブルーチンから構成されています．

(1)　単精度実数を倍精度実数に変換するルーチン

アドレス	＄ＢＣＤ５
機　能	ＦＡＣ１の内容を単精度実数から倍精度実数に変換する．
レジスタ	Ａ，Ｘ
入力情報	ＦＡＣ１　＝単精度実数
復帰情報	ＦＡＣ１　＝倍精度実数 （００１７）＝倍精度型を示す数値（＄０８）
ＷＯＲＫ	（７８～７Ｂ），（００１７）

解　説　このルーチンは４つの変換ルーチンの中で最も簡単です．以下その概略を述べます．

(i)　単精度型では未使用の（７８～７Ｂ）を初期化します．

(ii)　＄００１７番地に＄０８を入れます．

(2) 整数を単精度実数に変換するルーチン

アドレス	＄ＢＣＦＦ
機　　能	ＦＡＣ１を整数から単精度実数に変換する.
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　　＝整数
復帰情報	ＦＡＣ１　　＝単精度実数 （００１７）＝単精度実数を示す数値（＄０４）
ＷＯＲＫ	ＦＡＣ１ （００８Ｃ）＝保護バイト　５－２－３参照
ＳＵＢ	＄Ｂ３ＡＤ→符号を判定して正規化ルーチンを呼ぶ

解　　説　以下にこのルーチンの簡単なアルゴリズムを示します．図５・２・２を御覧下さい．

(i)　整数データを１バイトずつ上へシフトします（①の図）．

(ii)　＄００１７番地を＄０２（整数型）から＄０４（単精度型）へ変えます．

(iii)　＄９０を＄００７４番地（指数部）に代入します．

※　ここでなぜ指数部に＄９０を入れるか述べてみます．②の図を御覧下さい．この状態で＄００７６番地の b_0（整数の最下位）は 2^0 の位になります．ここで実数は　Ｘ＊2^n の形で表されることを思い出して下さい．ＦＡＣ１の指数部に入るのは（＄８０＋ｎ）です．2^0 の位のことを考えると整数の１つ上のビット，つまりＸの小数点の１つ上のビットは２の１６(＄１０)乗の位になります．この値がｎとなります．ですから指数部に（＄８０＋＄１０）を入れれば実数型に変換できるわけです．

(iv)　数値が負であれば仮数部のＮＥＧをとります（整数は２の補数表現）．

(v)　正規化ルーチン（５－２－３参照）を実行します．

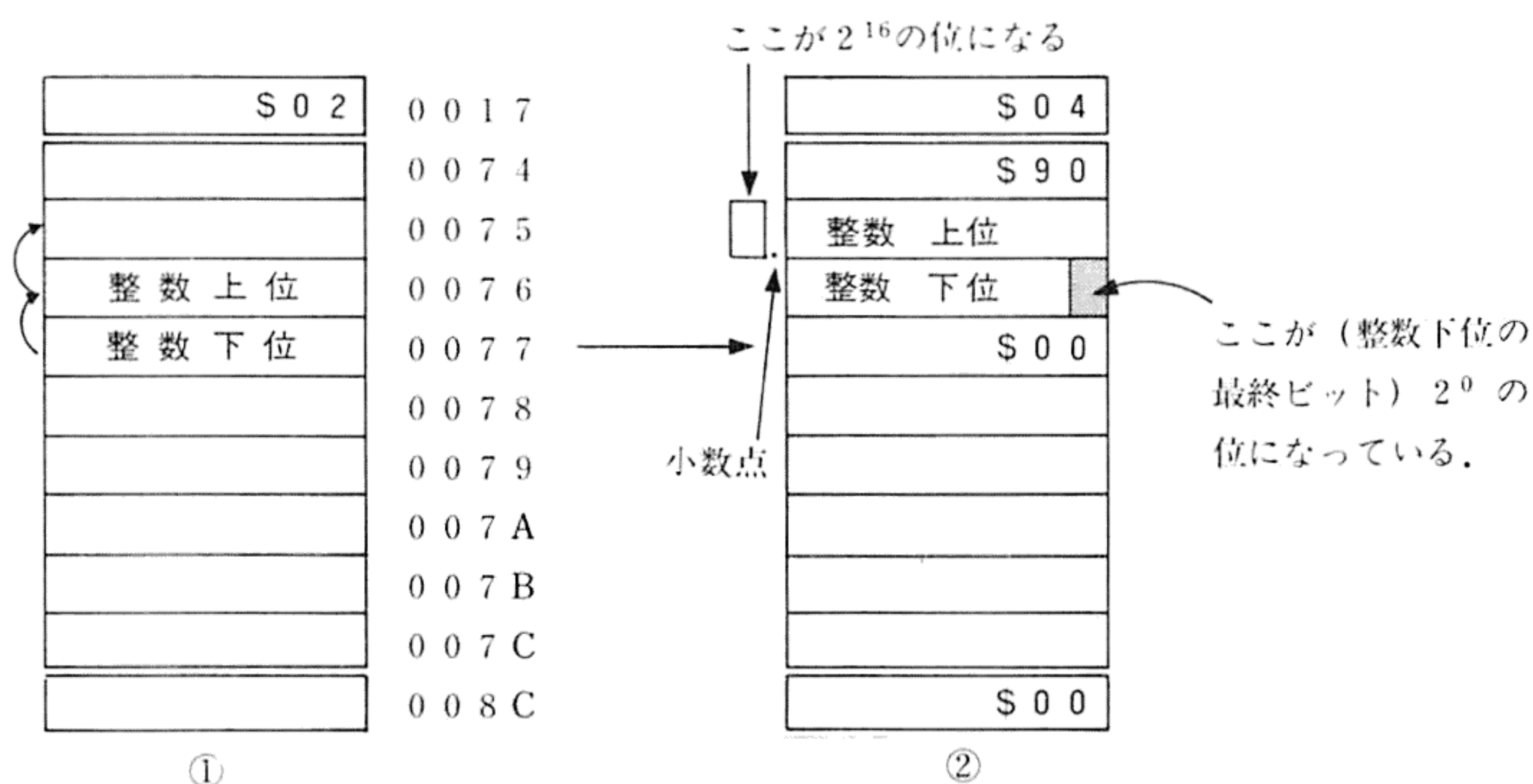
※　例えば，＄００８１などの整数は仮数部の一番最初に１が立ちませんので正規化ルーチンを行います．

図５・２・２　整数型を単精度型にする

①	アドレス	②
$02	0017	$04
	0074	$90
	0075	整数　上位
整数上位	0076	整数　下位
整数下位	0077	$00
	0078	
	0079	
	007A	
	007B	
	007C	
	008C	$00

ここが 2^{16} の位になる

小数点

ここが（整数下位の最終ビット）2^0 の位になっている．

(3)　単精度実数を整数に変換するルーチン

アドレス	$BC87
機　　能	FAC１の内容を単精度実数から整数に変換する．
レジスタ	A，B
入力情報	FAC１　＝単精度実数
復帰情報	FAC１　＝整数 （0017）＝整数を示す値（$02）
WORK	FAC１，（0081）
SUB	$B435→FACの符号を調べ切り捨てを行う
終了条件	FAC１の値が－32767～＋32767を越えると$B044番地へ飛んでError（Overflow）を発生する．

解　　説　ほぼ以下のアルゴリズムを行っています．

(i)　指数部が＄９０以上のとき（ＦＡＣ１の絶対値が３２７６８以上のとき）はError（Overflow）を発生します．

(ii)　指数部から＄９８を引きます．その値をＢレジスタへ入れて切り捨てのルーチン（５－２－４参照）を呼びます．

※　指数部から＄９８を引く理由

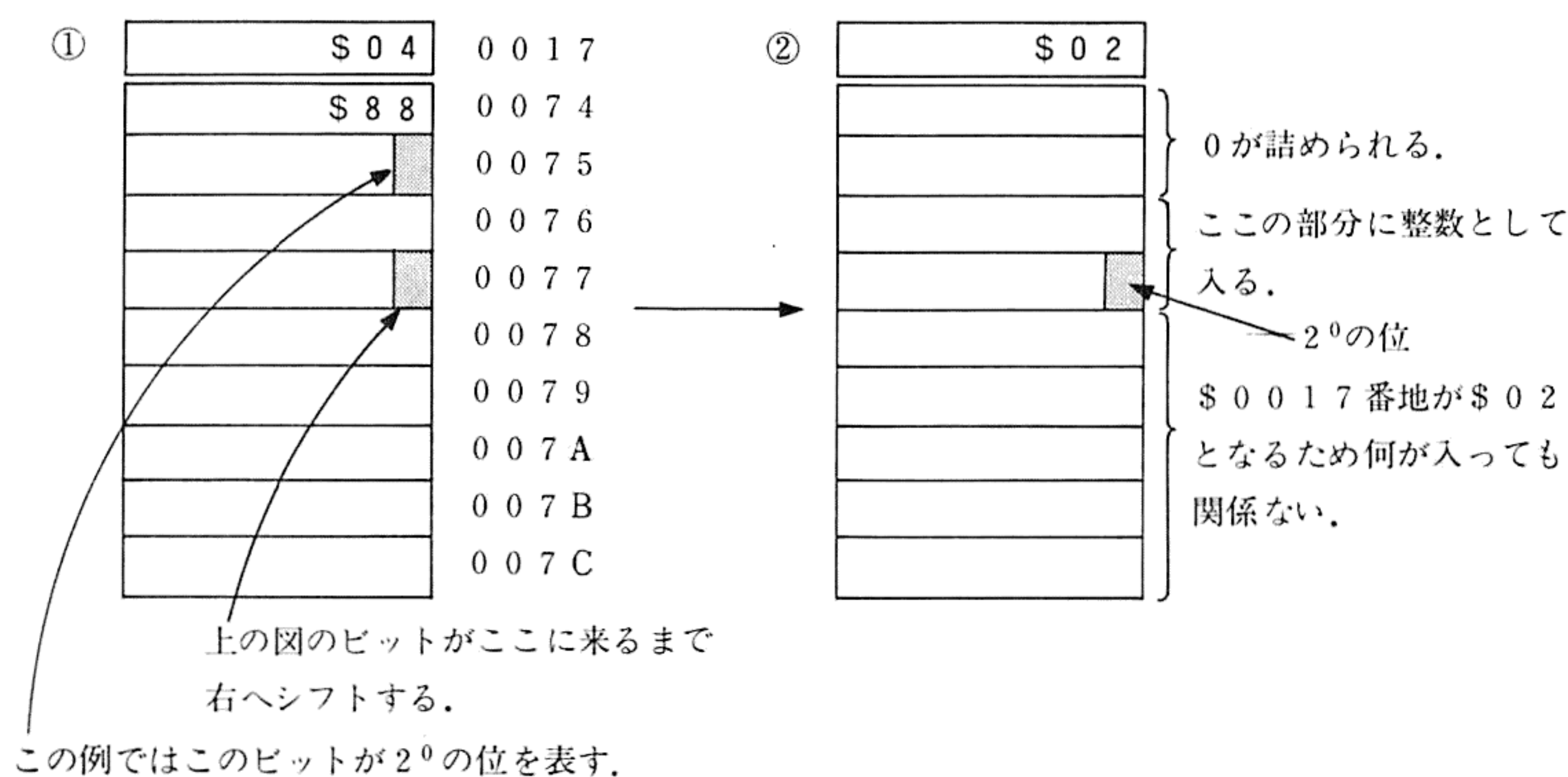
図５・２・３を御覧下さい．①のように指数部に＄８８が入っていたとします．＄００７５番地の b_7 は 2^7 の位に相当しますから 2^0 の位は＄００７５番地の b_0 のところになります．ですからこの 2^0 の位が＄００７７番地の b_0 に来るまで右へシフト（切り捨て）すれば（７６：７７）に整数の絶対値が入

ります．この場合右へシフトするのは＄１０回です．この値を得るのが（指数部－＄９８）です．

(iii) 数値が負のときは（７６：７７）のＮＥＧをとります．

(iv) ＄００１７番地へ＄０２を代入します．

※ このルーチンではＦＡＣ１に－３２７６８を入れても Error を発生します．

図５・２・３　単精度型を整数型にする（例）

(4) 倍精度実数を単精度実数に変換するルーチン

アドレス	＄ＢＣＥＥ
機　能	ＦＡＣ１の内容を倍精度型から単精度型に変換する．
レジスタ	Ａ，Ｘ
入力情報	ＦＡＣ１　＝倍精度実数
復帰情報	ＦＡＣ１　＝単精度実数 （００１７）＝単精度型を示す数値（＄０４）
WORK	ＦＡＣ１，（００１７）
ＳＵＢ	＄ＡＦＥＣ（下の(iv)の所を実行している）

解　説

(i) ＄００１７番地に＄０４（単精度型）を代入します．

(ii) ＦＡＣ１が０のときは何もせずにリターンします．

(iii) ＄００７８番地の内容を保護バイトへ移します．

(iv) 丸めルーチンを実行します．

※ (iii)と(iv)では，＄００７８番地以下を四捨五入すると考えて下さい．

5－2－6　数値を転送するルーチン

Ｆ－ＢＡＳＩＣ内で数値を転送するものには，２種類あります．一つはＦＡＣからＦＡＣへ移すものです．もう一つはＦＡＣと変数テーブル間での転送をするものです．

ＦＡＣ相互の間での転送は比較的容易ですが，少々問題があるのが変数テーブルとＦＡＣ間での数値の転送です．

変数テーブルとは，変数エリア内の数値を置く場所のことを指しますが，ここに格納される数値は必要最小限のバイト数で情報をたくわえるようにされているため，ＦＡＣ上の格納形式とは少々異なっています．

以下この形式の数値のことを変数テーブル用数値とします．

(1)　変数テーブルとＦＡＣとのデータ格納の違い

(ｉ)整数

２－２－１を御参照下さい．

(ii)単精度実数（ＦＡＣ１を例とします）

ＦＡＣに入る演算の結果は必ず正規化（５－２－３参照）されます．つまり仮数部の最上位バイトのb_7は必ず１が立ちます．ですからこの１ビットに他の情報を入れたとしても後でこのb_7を１にすれば元の数値にもどります．ここに＄００７Ｃ番地の１ビットしか使っていない符号ビットを代入します．このようにしたのが，変数テーブル用数値です（図５・２・４参照）．

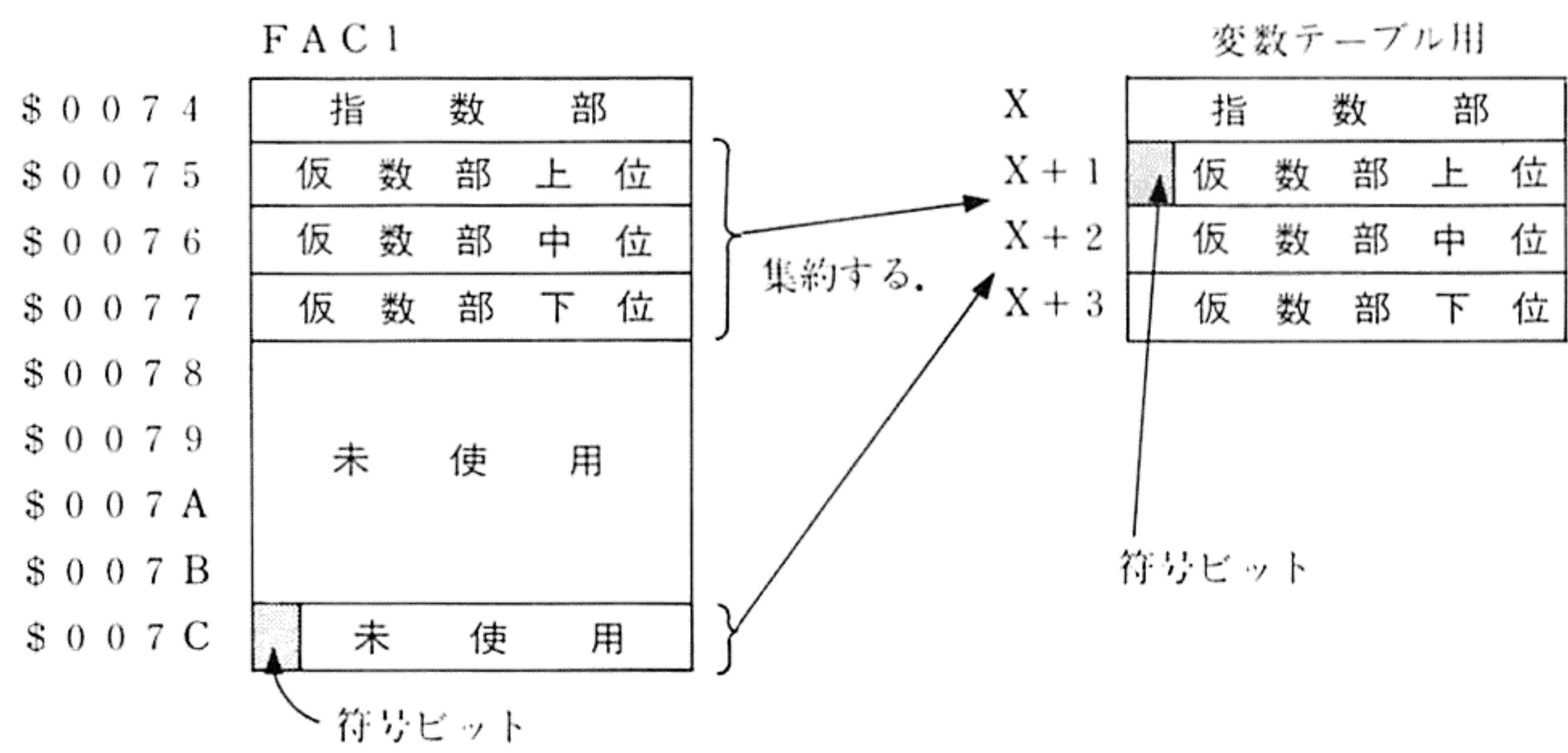
(iii)倍精度実数

この場合も単精度型の場合とほぼ同じです（ただ単精度型では未使用の箇所を仮数部下位の下へ保存します）．

図５・２・４　単精度実数の格納

	ＦＡＣ１
＄００７４	指　数　部
＄００７５	仮数部上位
＄００７６	仮数部中位
＄００７７	仮数部下位
＄００７８	未　使　用
＄００７９	
＄００７Ａ	
＄００７Ｂ	
＄００７Ｃ	符号ビット／未　使　用

集約する．

	変数テーブル用
Ｘ	指　数　部
Ｘ＋１	符号ビット／仮数部上位
Ｘ＋２	仮数部中位
Ｘ＋３	仮数部下位

(2) ＦＡＣからＦＡＣへの数値の転送

(ⅰ)ＦＡＣ１からＦＡＣ２へ転送するルーチン

アドレス	＄Ｂ３８０
機　能	ＦＡＣ１からＦＡＣ２へ数値を転送する．
レジスタ	Ａ，Ｘ
入力情報	ＦＡＣ１　　＝数値
	（００１７）＝転送したい数値の数値型を示す値
復帰情報	ＦＡＣ２　　＝ＦＡＣ１の値
ＷＯＲＫ	ＦＡＣ１，ＦＡＣ２
ＳＵＢ	＄ＢＣＢ３→数値型判断ルーチン５－２－１参照

(ⅱ)ＦＡＣ２からＦＡＣ１へ転送するルーチン

アドレス	＄Ｂ３５Ｆ
機　能	ＦＡＣ２から数値をＦＡＣ１へ転送する．
レジスタ	Ａ，Ｘ
入力情報	ＦＡＣ２　　＝数値
	（００１７）＝転送したい数値の数値型を示す値
復帰情報	ＦＡＣ１　　＝ＦＡＣ２の値
ＷＯＲＫ	ＦＡＣ１，ＦＡＣ２

(3) 変数テーブル（メモリ）とＦＡＣ間の数値の転送

(ⅰ)メモリから変数テーブル用数値をＦＡＣ２へ転送するルーチン

アドレス	＄Ｂ１ＢＡ
機　能	Ｘレジスタの示す番地から代入されている変数テーブル用数値を，ＦＡＣ２へ代入します．
レジスタ	Ａ，Ｂ
入力情報	Ｘレジスタ　＝数値の入っている箇所の先頭アドレス
	（００１５）＝単精度型のとき＄００，倍精度型のとき＄００以外
復帰情報	ＦＡＣ２　　＝変数テーブル用数値をＦＡＣ用に変換して代入
	Ａレジスタ　＝数値代入後の＄００８２番地の値
	Ｂレジスタ　＝＄００７４番地の値（ＦＡＣ１の指数部）
	（００８Ｂ）＝変数テーブル用数値とＦＡＣ１の符号によって b_7 を次のように決定する

ＦＡＣ１	変数テーブル	＄８Ｂ番地b_7
＋	＋	0
－	＋	1
－	－	0
＋	－	1

WORK　ＦＡＣ２，（００８Ｂ），ＦＡＣ１

解　説　このルーチンは，数値が変数テーブル用数値の格納の形式に従って代入されていれば，変数テーブル上になくても任意のメモリから数値をＦＡＣ２に代入することができます．

なおこのサブルーチンは実数型の数値しか扱えません．

（ⅱ）メモリから変数テーブル用数値をＦＡＣ１へ転送するルーチン

アドレス　＄Ｂ３０１

機　能　Ｘレジスタの示す番地から入っている変数テーブル用の数値を，ＦＡＣ１へ代入する．

レジスタ　Ｂ，Ｕ

入力情報　Ｘレジスタ　＝数値の入っている箇所の先頭アドレス
（００１７）＝転送したい数値の型を示す値

復帰情報　ＦＡＣ１　＝変数テーブル用数値をＦＡＣ用に変換して代入

WORK　ＦＡＣ１，（００１７）

解　説　このサブルーチンはすべての数値型で使用可能です．

（ⅲ）ＦＡＣ１からメモリに変数テーブル用数値を転送するルーチン

アドレス　＄Ｂ３２Ｂ，＄Ｂ３３０，＄Ｂ３３７

機　能　＄Ｂ３２Ｂ：単精度実数のＦＡＣ１の内容を＄００６３番地からの４バイトに変数テーブル用数値で代入する．

＄Ｂ３３０：単精度実数のＦＡＣ１の内容を＄００５Ｆ番地からの４バイトに変数テーブル用数値で代入する．

＄Ｂ３３５：ＦＡＣ１の内容をワークレジスタ（５Ａ：５Ｂ）の示す番地から変数テーブル用数値として代入する（ＬＥＴ文処理用）．

＄Ｂ３３７：ＦＡＣ１の内容をＸレジスタの示す番地から変数テーブル用数値として代入する．

レジスタ　A，X，U

入力情報　＄B32B，＄B330のとき

（0017）　＝＄04（単精度型）

FAC1　＝単精度実数

＄B335のとき

（5A：5B）　＝変数実効アドレス（代入用）

（0017）　＝FAC1の数値型を示す値

FAC1　＝数値

＄B337のとき

（0017）　＝FAC1の数値型を示す値

FAC1　＝数値

Xレジスタ　＝数値を転送するメモリの先頭アドレス

WORK　FAC1，（0017）

5－2－7　数値→文字列変換ルーチン

アドレス	＄B620
機　能	FAC1の数値を文字列にする．
入力情報	FAC1　　＝数値
	(0017)＝FAC1の示す数値型
復帰情報	Xレジスタ　　＝文字列の先頭アドレス（＄0541）
	(0541～)＝文字列
WORK	(00BA)　＝FORMAT指定子
	(0063)　＝小数点以下の桁数
	(0064)　＝コンマカウンタ
	(0069)　＝指数部操作用レジスタ
	(007D)　＝符号フラグ
	(0015)　＝倍精度フラグ
	(8D：8C)＝Xレジスタの保存
SUB	＄BCB3→数値型判断ルーチン　5－2－1参照
	＄B3C8→符号判定ルーチン　5－2－2参照
	＄BDC9→符号反転ルーチン

解　説　FAC1の数値をその型にしたがって＄0541番地から文字列として格納します．この際単精度型では7桁目，倍精度型では17桁目を四捨五入して数値を文字列に変換します．ですから単精度型の実数では最終の2ビット～3ビットは無視される結果になります．ここの数値を得たいときは単精度実数をCDBLなどで倍精度型にすると求まります．一般にCDBLを単精度実数に対して実行しても無意味な数値が6桁以降に出力されるように思われがちですが，実はこのように有効な数値が出力（1～2桁）されているのです．

5−2−8　文字列→数値変換ルーチン

アドレス	＄Ｂ５０６，＄Ｂ５０Ｂ
機　　能	＄Ｂ５０６：倍精度型を基本にして数値を求める．
	＄Ｂ５０Ｂ：整数型を基本にして数値を求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	（Ｄ９：ＤＡ）＝文字列の先頭アドレス−１
復帰情報	ＦＡＣ１　　　＝文字列を変換した数値
	（００１７）　＝ＦＡＣ１の型を表わす数値
	（Ｄ９：ＤＡ）＝数値を表わす文字列の次のアドレス
ＷＯＲＫ	ＦＡＣ１
	（００６３）　＝小数点以下の桁数
	（００６４）　＝小数点フラグ
	（００６９）　＝指数部操作用レジスタ
	（００１５）　＝倍精度フラグ
	（００１７）　＝ＦＡＣ１の数値の型
	（Ｄ２～ＤＤ）＝汎用読み込みルーチン
ＳＵＢ	省略

解　説　＄Ｂ５０６の場合　＆，＆０，＆Ｈ，！，Ｅなどの指定がないかぎり倍精度型で数値が得られます．整数型の％があっても整数型にはなりません．
＄Ｂ５０Ｂの場合　整数型を基本にして数値を求めたいときに使います．＆，＆０，＆Ｈ，！，Ｅ，＃などの指定があればそれに従います．また整数の範囲を越えた場合は単精度型（それでも収まらなければ倍精度型）で数値が求まります．

ただし両方ともＥが「ＥＬＳＥやＥＱＶのＥ」の場合は単精度型に変換せず，Ｅが現れる前までの文字列を数値とし，Ｅのアドレスを（Ｄ９：ＤＡ）に入れてすぐリターンしてしまいます．

ＥＸＥＣなどでこのルーチンを使用するときは必ず（Ｄ９：ＤＡ）を一時退避して下さい（暴走することがあります）．

5－3　2項演算

5—3—1　実数の加算を行うルーチン

アドレス	＄ＡＦ１Ａ
機　　能	メモリ（数値を変数テーブル用数値で格納）＋ＦＡＣ１を実行する．

メモリとＦＡＣ１の数値型は同一でなければならない．

レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　＝加算する数値(単精度実数，あるいは倍精度実数) （００１５）＝倍精度型のとき＄００以外，単精度型のとき＄００ Ｘレジスタ　＝メモリ（変数テーブル用数値で数値が格納されている箇所）の先頭アドレス メモリ　＝ＦＡＣ１と同じ数値型のデータを変数テーブル用数値であらかじめメモリ上に格納（被加算数値）
復帰情報	ＦＡＣ１　＝加算された結果
ＷＯＲＫ	ＦＡＣ１，ＦＡＣ２ （００１５）＝倍精度フラグ （００８Ｂ）＝表５・３・１参照 （００８Ｃ）＝保護バイト
ＳＵＢ	＄Ｂ１ＢＡ→メモリをＦＡＣ２へ　５－２－６参照 ＄Ｂ３５Ｆ→ＦＡＣ２をＦＡＣ１へ　５－２－６参照 ＄ＡＦ０５→Ｂレジスタの内容だけ切り捨て ＄Ｂ０７４→Ｂレジスタの内容だけ切り捨て
終了条件	1.7014×10^{38}～-1.7014×10^{38}の範囲外のとき Error（Overflow）を発生する．

解　　説　このルーチンはメモリ（変数テーブル用数値で代入されている）にＦＡＣ１の値を加えます．その際数値の型は両方とも同じ型でなければなりません．

一般的な加算のアルゴリズムは以下のようになります．

Ａ．２つの数値の指数部が同じになるように調整

Ｂ．仮数部の加算

Ｃ．正規化

図５・３・１を御覧下さい．１０進数の加算を示しました．Ａの操作が①の操作，Ｂの操作が②の操作，Ｃの操作が③の操作に相当します．

図5・3・1　10進数加算

$1.5\times10^3+1.3\times10^5$ を実行する.

① 桁を合わせる.
```
    1500
+130000
```

②加算をする.
```
    1500
+130000
 131500
```

③指数型式にする.

$131500=1.315\times10^5$

F－BASICでは以上のことを次のように実行しています.

(1)　メモリをFAC２へ移します. ですからこの後ではFAC２＋FAC１を実行しています. また, ここでは＄００８Ｂ番地のb_7を下の表のようにセットします.

FAC１の符号	メモリの符号	＄８Ｂ番地b_7
＋	＋	0
＋	－	1
－	＋	1
－	－	0

表5・3・1

(2)　FAC１＝０ならFAC２が答えになりますからFAC２をFAC１へ代入して終ります. 他のときは(3)を実行します.

(3)　FAC２＝０ならFAC１が答えになりますから何もせずに終ります. 他のときは(4)を実行します.

(4)　FAC２の指数部からFAC１の指数部を引きます. この結果によってFAC１の桁をFAC２に合わせるか, FAC２の桁をFAC１に合わせるかの場合分けを以下のようにします（ここでは前に述べたAを行っています).

(i)　FAC２の指数部がFAC１の指数部より大きいとき

FAC１の指数部がFAC２の指数部と同じになるまでFAC１の仮数部を右へシフトします（FAC１の指数部をFAC２に合わせています).

（００７C)に(００８A）の値を代入します（結果の符号部をFAC２の値とします). 次に（００８２）の値を(００７４）に代入します（結果の指数部をFAC２の値にします).

(ii)　ＦＡＣ１の指数部がＦＡＣ２の指数部より大きいとき

ＦＡＣ２の指数部がＦＡＣ１の指数部と同じになるまで仮数部を右へシフトします（ＦＡＣ２の指数部をＦＡＣ１の指数部に合わせています）．

結果の符号部と指数部はＦＡＣ１のものとします．

(iii)　ＦＡＣ１の指数部とＦＡＣ２の指数部が等しいとき　何もしません．

(5)　(1)で求めた＄００８Ｂ番地の内容と(4)の(i)，(ii)，(iii)の条件により場合分けをします．

(i)　＄００８Ｂ番地の b_7 が０のとき

仮数部の足し算をします．桁あふれが出た場合は，結果の指数部に１を加えて仮数部を右へ１ビットずらします

(ii)　＄００８Ｂ番地の b_7 が１のとき

㋑　(4)で(i)のとき（ＦＡＣ２の指数部がＦＡＣ１の指数部より大きいとき），

ＦＡＣ１の仮数部のＮＥＧ（２の補数）をとり，それにＦＡＣ２の仮数部を加えます（ＦＡＣ２の仮数部からＦＡＣ１の仮数部を引く（例２参照））．

㋺　(4)で(ii)のとき（ＦＡＣ１の指数部がＦＡＣ２の指数部より大きいとき）

ＦＡＣ２の仮数部のＮＥＧ（２の補数）をとり，それにＦＡＣ１の仮数部を加えます（ＦＡＣ１の仮数部からＦＡＣ２の仮数部を引く）．

㋩　(4)で(iii)のとき（ＦＡＣ１の指数部とＦＡＣ２の指数部が等しいとき）

ＦＡＣ２の仮数部のＮＥＧを取り，ＦＡＣ１に加えます（ＦＡＣ１の仮数部からＦＡＣ２の仮数部を引きます）．ＦＡＣ２の仮数部がＦＡＣ１の仮数部より大きいときは，符号のバイト（００７Ｃ）を反転させて，引いた結果のＮＥＧをとります（例３参照）．

(6)　正規化します．

一応のアルゴリズムは述べましたが，これではまだよくわからないと思いますので，２～３例をあげて説明します．簡単にするため仮数部を４ビットとします．

〈例１〉　(5)の(i)の場合（ＦＡＣ１とメモリが同符号）

(1)　$\text{FAC1} = 0.1010 \times 2^2$

　　$\text{FAC2} = 0.1110 \times 2^5$（メモリの値）

ＦＡＣ２の指数部＝５＞２＝ＦＡＣ１の指数部　ですから指数部をＦＡＣ２に合わせます．

ＦＡＣ１＝０．０００１０１０×2^5

ＦＡＣ２＝０．１１１０×2^5

このとき結果の符号と指数部はＦＡＣ２のものとなりますから結果は以下のようになります．

０．１１１１×2^5（４ビット以下は丸める）

⑵　ＦＡＣ１＝０．１０１０×2^2

ＦＡＣ２＝－０．１１１０×2^5（メモリの値）

⑴の場合と同様に符号はＦＡＣ２の符号部が決定しますから

－０．１１１１×2^5となります．

⑶　ＦＡＣ１＝０．１０１０×2^5

ＦＡＣ２＝０．１０００×2^5（メモリの値）

今度は桁あふれする場合です．仮数部，指数部は以下のようになります．

１．００１０×2^5

最初の１は桁あふれになります．ですから下のように直します．

０．１００１×2^6

〈例２〉　⑸の(ii)の㋑の場合　ＦＡＣ１の指数部＜ＦＡＣ２の指数部

ＦＡＣ１とＦＡＣ２が異符号の場合

ＦＡＣ１＝０．１０００×2^2

ＦＡＣ２＝－０．１０００×2^3（メモリの値）

この場合ＦＡＣ２の絶対値は必ずＦＡＣ１の絶対値より大きくなりますから符号は必ずＦＡＣ２の値になります．また仮数部はＦＡＣ２からＦＡＣ１を引いた値になります．つまり（－４）＋２を－（４－２）としているのと同じことなります．それでは実際に行ってみることにします．ＦＡＣ１の指数部をＦＡＣ２に合わせます．

ＦＡＣ１＝０．０１００×2^3

ＦＡＣ２＝－０．１０００×2^3

ＦＡＣ１の仮数部のＮＥＧをとります．

ＦＡＣ１の仮数部のＮＥＧ＝０．１１００

（初めの０はＦＡＣ１内部にありませんので値は変わりません）

ＦＡＣ２の仮数部とＦＡＣ１の仮数部（ＮＥＧしたもの）を加えます．

０．０１００　　実際には桁あふれ（最初の０の所に１が立つ）がありま

すが，ＮＥＧをとって加えたときは無視します．
指数部・仮数部ともにＦＡＣ２の値になりますから結果は
-0.0100×2^3　と求まります．さらに正規化して
-0.1000×2^2　となります．

(ロ)の場合も同様にして求めることができます．

〈例３〉 (5)の(ii)の(ハ)の場合　ＦＡＣ１とＦＡＣ２が異符号で指数部が等しいとき

(1)　ＦＡＣ１の仮数部＞ＦＡＣ２の仮数部のとき
ＦＡＣ１$=0.1100\times2^3$
ＦＡＣ２$=-0.1000\times2^3$（メモリ）
ＦＡＣ２の仮数部のＮＥＧをとります．
０．１０００　（最初の０はＦＡＣの内部にありません）
ＦＡＣ１の仮数部に足します．
０．０１００　（桁上がりを無視します）
符号をＦＡＣ１の符号（このときもＦＡＣ１の絶対値がＦＡＣ２より大きいため）に決定します．
0.0100×2^3　を正規化して
0.1000×2^2　となります．

(2)　ＦＡＣ１の仮数部＜ＦＡＣ２の仮数部のとき
ＦＡＣ１$=-0.1000\times2^3$
ＦＡＣ２$=0.1100\times2^3$（メモリの値）
ＦＡＣ２の仮数部のＮＥＧ
０．０１００　（最初の０はＮＥＧには関係ありません）
ＦＡＣ１の仮数部とＦＡＣ２の仮数部を足します．
０．１１００
この値のＮＥＧをとります．
０．０１００　（この値が結果の仮数部となる）
符号はＦＡＣ１の反対つまりＦＡＣ２の値になります（ＦＡＣ２の絶対値がＦＡＣ１の絶対値より大きい）．結果は以下のようになります．
-0.0100×2^3　を正規化して
-0.1000×2^2　となります．

5－3－2　実数の減算を行うルーチン

アドレス	＄ＡＦ１１

機　能	メモリ(数値を変数テーブル用数値で格納)からＦＡＣ１を引く．

メモリとＦＡＣ１の数値型は同一でなければならない．

レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　＝減算する数値(単精度実数，あるいは倍精度実数)
	（００１５）＝倍精度型のとき＄００以外，単精度型のとき＄００
	Ｘレジスタ　＝メモリ（変数テーブル用数値で数値が格納されている箇所）の先頭アドレス
	メモリ　＝ＦＡＣ１と同じ数値型のデータを変数テーブル用数値であらかじめメモリ上に格納（被減算数値）
復帰情報	ＦＡＣ１　＝減算された結果
ＷＯＲＫ	ＦＡＣ１，ＦＡＣ２
	（００１５）＝倍精度フラグ
	（００８Ｂ）＝表５・３・１参照
	（００８Ｃ）＝保護バイト
ＳＵＢ	＄Ｂ１ＢＡ→メモリをＦＡＣ２へ　５－２－６参照
	＄ＡＦ１Ｄ→ＦＡＣ２からＦＡＣ１を引く
終了条件	結果が1.7014×10^{38}〜-1.7014×10^{38}の範囲外のときはError（Overflow）を発生する．

解　説　一般に減算は被減算数値をＸ，減算数値をＹとすると次のように加算で表すことができます．

$$X-Y=X+(-Y)$$

つまりＹの符号を変えて加算すればよいわけです．Ｆ－ＢＡＳＩＣではＸがメモリ，ＹがＦＡＣ１となっています．ですからメモリをＦＡＣ２へ移してから＄００７Ｃ番地（ＦＡＣ１の符合部）を反転させ，その後＄００８Ｂ番地を反転させ，加算ルーチンを呼び出せば減算を行います．

※　＄００８Ｂ番地を反転させるのは，表５・３・１より求まる＄００８Ｂ番地の値を，＄００７Ｃ番地を反転させる前の値によって求めているからです．

5－3－3　実数の乗算を行うルーチン

アドレス　＄Ｂ０ＥＥ

機　能　メモリ(数値を変数テーブル用数値で格納)とＦＡＣ１の乗算を実行する．メモリとＦＡＣ１の数値型は同一でなければならない．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　ＦＡＣ１　＝乗算する数値(単精度実数，あるいは倍精度実数)

（００１５）＝倍精度型のとき＄００以外，単精度型のとき＄００

Ｘレジスタ　＝メモリ（変数テーブル用数値が格納されている箇所）の先頭アドレス

メモリ　＝ＦＡＣ１と同じ数値型のデータを変数テーブル用数値であらかじめメモリ上に格納（被乗算数値）

復帰情報　ＦＡＣ１　＝乗算された結果

ＷＯＲＫ　ＦＡＣ１，ＦＡＣ２，ＦＡＣ３

（００１５）＝倍精度フラグ

（００８Ｂ）＝ここでは演算結果の符号バイト

（００８Ｃ）＝保護バイト

ＳＵＢ　＄Ｂ１ＢＡ→メモリをＦＡＣ２へ　５－２－６参照

＄Ｂ１ＤＦ→符号部と指数部の処理

＄Ｂ１３６→各桁の乗算を行う

＄Ｂ１３Ａ→同上

＄Ｂ２ＥＣ→ＦＡＣ３をＦＡＣ１へ

＄ＡＦ９７→正規化ルーチン　５－２－３参照

終了条件　結果が1.7014×10^{38}～-1.7014×10^{38}の範囲外のときError（Overflow）を発生する．

解　説　乗算の計算は被乗算数値をＸ，乗算数値をＹと下のようにおいて，

$X = 2^{n} * V$　　　$Y = 2^{m} * U$　　（ＶとＵは仮数部）

以下の計算を実行します．

$X * Y = 2^{n+m} * V * U$

Ｆ－ＢＡＳＩＣでは以上のことを次のように実行します．

(1) 演算結果の指数部と符号の決定

＄００７４番地（ＦＡＣ１の指数部）に（ｎ＋ｍ＋＄８０）を代入します．これは結果の指数部を求めています．

次に結果の符号部を求めるのですが，これはＦＡＣ２へ数値をメモリから代入する際に＄００８Ｂ番地が**表５・３・１**のように求まりますからこれを利用します．つまり二つの数値が異符号のとき負（b_7 に１が立つ），同符号のとき正（b_7 に０が代入される），と決定します．この値を＄００７Ｃ番地に入れます．これで符号と指数は求められたので次に仮数部を求め，これをＦＡＣ１の仮数部に代入すれば答えが求まります．

(2) 仮数部の計算

６８０９には，Ａレジスタの値を掛け合わせて，Ｄレジスタに答えを入れるＭＵＬという命令がありますから比較的楽に（１０進数の掛け算のように）計算できます．

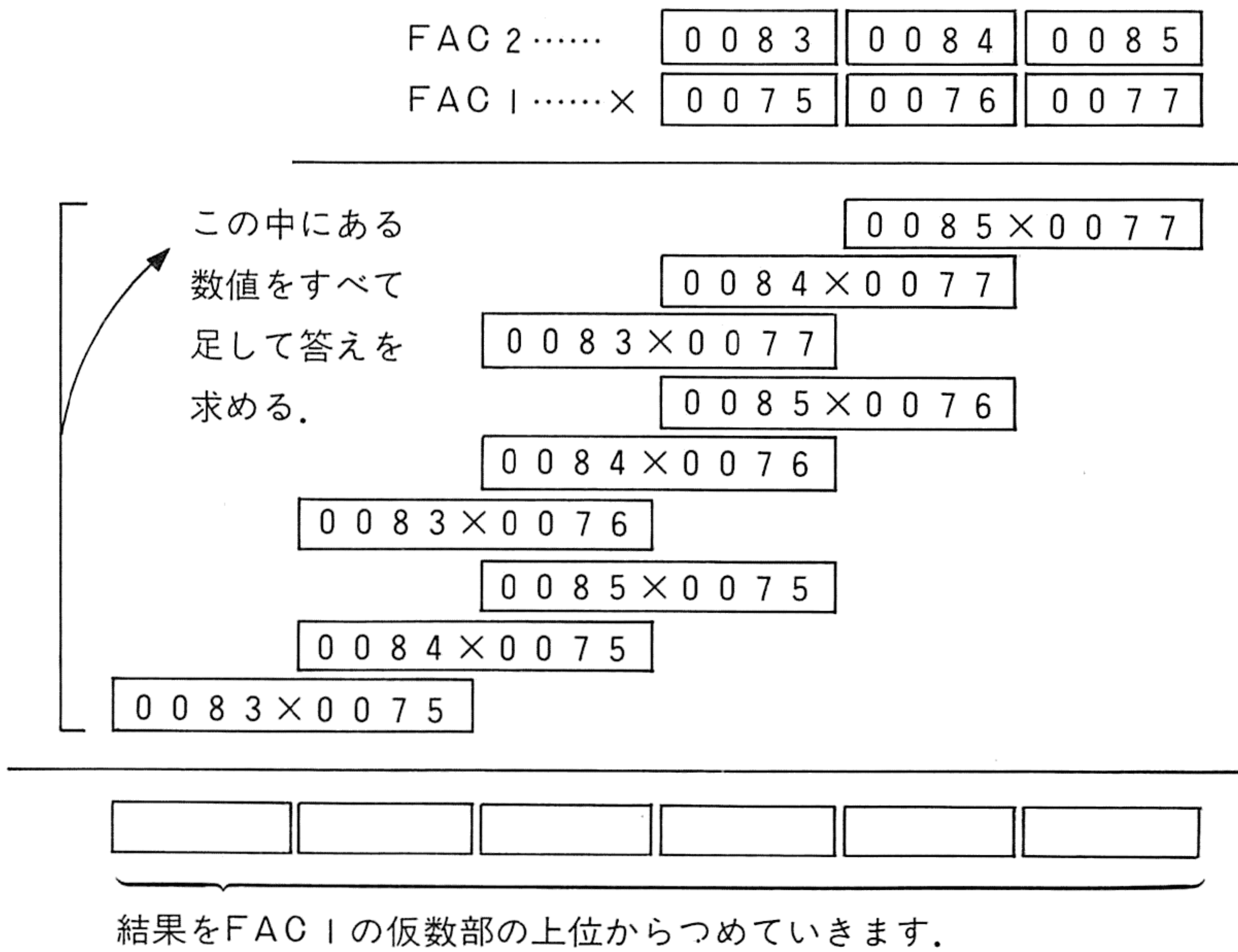
図５・３・２を見て下さい．１０進の計算の仕方をこのように書いてみます．次に**図５・３・３**を見て下さい．これが仮数部の演算です（ここでは単精度型を扱いました）．１つのバイトが１０進数の１桁に相当します．つまり３桁の算を行っているのと同じになるわけです．具体的には次のようにします．ＦＡＣ１の下位バイトにＦＡＣ２の下位バイトを掛け合わせてすぐ下に書きます(もちろんコンピュータではＦＡＣ３上に書く)．次にＦＡＣ１の中位バイトとＦＡＣ２の下位バイトを掛け合せ前の値の１桁ずらしたところに書きます．このようにして図のようにすべてのバイトを掛け合せていきます．その結果をすべて足し合わせて，ＦＡＣ１の仮数部の上位からつめていけば乗算の仮数部が求まるわけです．

図５・３・２　１０進数の掛け算

```
              8 7
            × 5 6
          ---------
              4 2 … 7 × 6
この値をすべ  4 8   … 8 × 6
て足します．  3 5   … 7 × 5
            4 0     … 8 × 5
          ---------
            4 8 7 2
```

図５・３・３　ＦＡＣの仮数部の乗算

```
FAC 2 ……    [0083][0084][0085]
FAC 1 ……×   [0075][0076][0077]
----------------------------------------
                       [0085×0077]
この中にある       [0084×0077]
数値をすべて   [0083×0077]
足して答えを       [0085×0076]
求める．       [0084×0076]
           [0083×0076]
               [0085×0075]
           [0084×0075]
       [0083×0075]
----------------------------------------
       [    ][    ][    ][    ][    ][    ]
```

結果をＦＡＣ１の仮数部の上位からつめていきます．

※ [　　]…１バイトを示します．
　 [　　　　]…２バイトを示します．

わくの中の数字はアドレスを示します．＄は省略しました．

5—3—4　実数の除算を行うルーチン

| アドレス | $B234

| 機　　能 | メモリ（数値を変数テーブル用数値で格納）をFAC1の値で割る．メモリとFAC1の数値型は同一でなければならない．

| レジスタ | A，B，X，U

| 入力情報 | FAC1　　＝除算する数値（単精度実数，あるいは倍精度実数）
（0015）＝倍精度型のとき$00以外，単精度型のとき$00
Xレジスタ　＝メモリ（変数テーブル用数値が格納されている箇所）の先頭アドレス
メモリ　　＝FAC1と同じ数値型のデータを変数テーブル用数値であらかじめメモリ上に格納（被減算数値）

| 復帰情報 | FAC1　　＝減算された結果

| WORK | FAC1，FAC2，FAC3
（0015）＝倍精度フラグ
（008B）＝結果の符号
（008C）＝保護バイト

| SUB | $B1BA→メモリをFAC1へ　5－2－6参照
$B1DF→指数部及び符号の演算
$B203→Error (Overflow) 発生
$B221→Error (Division By Zero) 発生

| 終了条件 | (1)結果が1.7014×10^{38}〜－1.7014×10^{38}の範囲外のときError（Overflow）を発生する．
(2)FAC1が0のとき，Error(Division By Zero)を発生する．

| 解　　説 | FAC1の値をY，メモリの値をXとし，X，Yを下のようにおいたとき

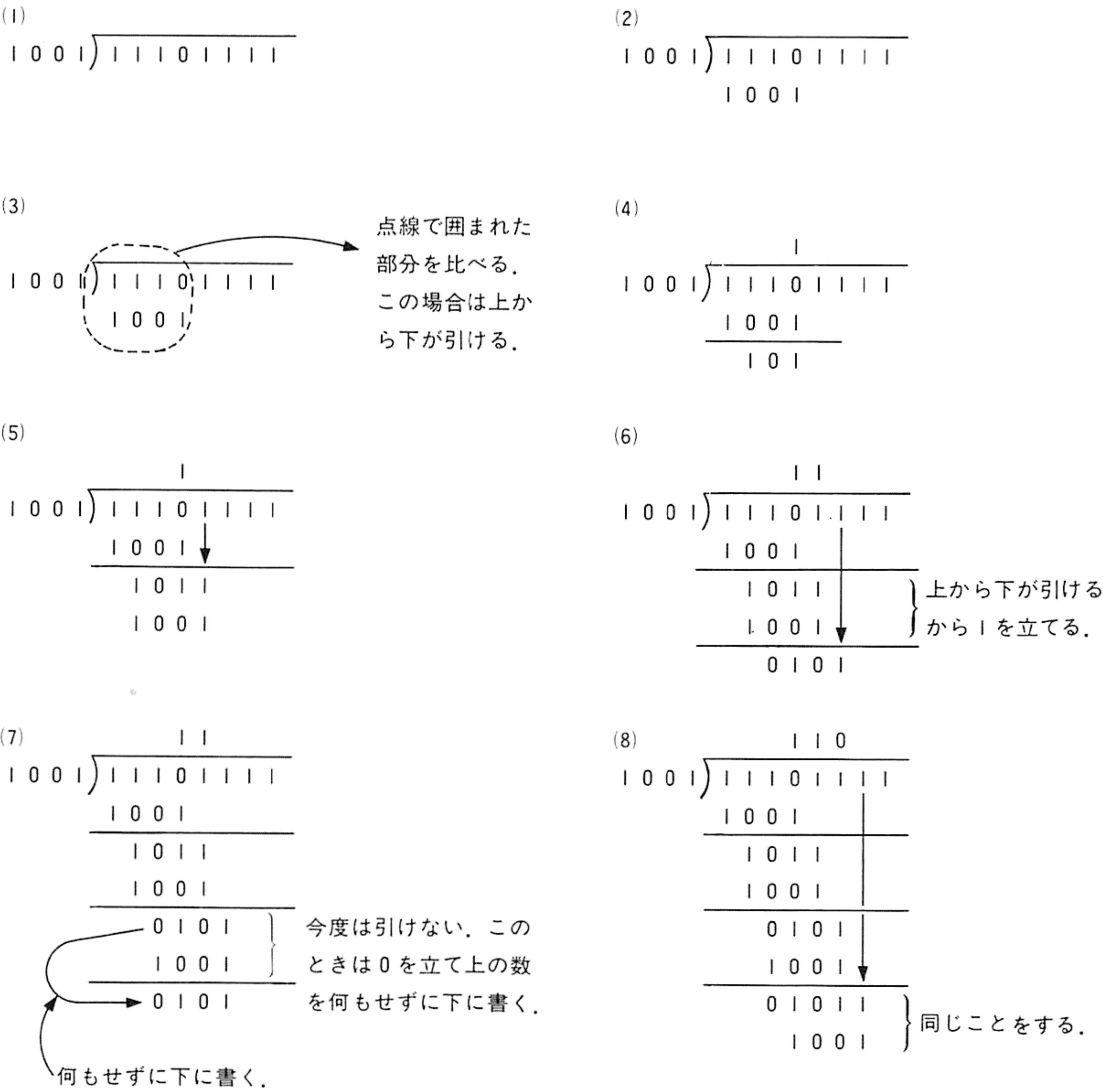
$X=2^{n}*V$　　　　$Y=2^{m}*U$　（UとVは仮数部）

以下の公式を用いて求めます．

$$\frac{X}{Y}=2^{n-m}*\frac{V}{U}$$

F－BASICでV／Uを計算するのに引き戻し法（実際には少し違う）という演算方法を用いています．この方法を，簡単にするため8ビットの数を4ビットで割る場合について図5・3・4を使って説明します．

(1)のように割られる数（１１１０１１１１）と割る数（１００１）を１０進数の割り算のように書いてみます．次に(2)のように割る数を左から割られる数の桁に合わせてその下に書きます．次に(3)の点線で囲まれた部分の上の数と下の数を比べます．上の数から下の数が引ける（上の数の方が大きい）ときは最終桁の上に１をたてます．引けないときは０を書きます．この場合は引けますから(4)のように１を立てます．そして引いた結果を前に書いた数値の下へ書きます．次に

図５・３・４　除算の方法

(1)

(2)

(3)

点線で囲まれた部分を比べる．この場合は上から下が引ける．

(4)

(5)

(6)

上から下が引けるから１を立てる．

(7)

今度は引けない．このときは０を立て上の数を何もせずに下に書く．

何もせずに下に書く．

(8)

同じことをする．

※　数値はすべて符号なし２進数です．

除算される数の１ビットを(5)のように下へずらし，引いた結果の横へ置きます．ここでまた(3)と同様にして上の数と下の数を比較します．今度も引けますので(6)のように１を立て，引いた答えをまた下に書きます，そしてまた同じように除算される数の１ビットを下へずらし，引いた結果の横へ書きます．そして(7)のようにしますが，今度は上の数から下の数を引くことができません．ですからこの上の数をそのまま結果に書き，答えを求める所に０を立てます．このようなことをくり返して行えば答が求まります．最終的には**図５・３・５**のようになります．

図５・３・５　除算の結果

```
1001=9
11101111=239
11010=26
101=5

239÷9=25……5
```

```
           11010
      ___________
 1001) 11101111
       1001        ←引ける
       ----------
        1011
        1001       ←引ける
        ---------
         0101
         1001      ←引けない
         --------
          1011
          1001     ←引ける
          -------
           101
          1001     ←引けない
          -------
           101     余り
```

では次にＦ－ＢＡＳＩＣ内部での演算を考えてみましょう．

(1)　結果の指数部を（ｎ－ｍ＋＄８０＋＄１）と置きます（※１）．結果の符号を**表５・３・１**の＄００８Ｂ番地の値にします．指数部を判断して結果が$1.7014\times10^{38}\sim1.7014\times10^{38}$の範囲外になるときはOverflowを発生します．

(2)　Ｙ（ＦＡＣ１）が０の場合は，Division By Zero を発生します．

(3)　仮数部を求めます．

ここで求める仮数部を　Z_1．Z_2　Z_3　Z_4　……とします（※２）．

Ｖ（メモリの仮数部），　Ｕ（ＦＡＣ１の仮数部）を比較して２通りに分けます．

(i)　Ｖ＞＝Ｕ（**図５・３・４**で引ける場合）

Z_i　＝１（商の所へ１を立てる）

Ｖ＝Ｖ－Ｕとおく（**図５・３・４**の(4)を実行）．　(4)へ飛ぶ．

(ii)　Ｖ＜Ｕ　（**図５・３・４**で引けない場合）

Z_i　＝０　　　　　　(4)へ飛びます．

Z_iが全部求められたときは(3)で終ります．

(4)　Ｖ＝２＊Ｖを実行します．

ｉ＝ｉ＋１として(3)に飛びます．

（※１）(※２）について

（※１）で＄１を足す理由は次の通りです．実は（※２）ではZ_1．Z_2　Z_3　でなく，０．Z_1　Z_2　Z_3　のように求めなければ，ＦＡＣに代入したときに１桁分ずれてしまいます．これを補正するのが＄１を足すことなのです．例えば　０．１を０．１で割ると答えは１．０となりますがこれをそのままＦＡＣの仮数部につめてしまうとコンピュータは１．０を０．１と判断してしまいます．この一桁分のずれをなくすのが＄１ということなのです．

※　引き戻し法について

大型コンピュータなどではＶとＵを比較するところでＶ－Ｕを行っています．その結果から引けるかどうかを判断します．ここで引けないときにはもとのＶを求めるのにＶ－Ｕ＋ＵというようにＵを足し直すわけです．つまり引いてからもとに戻すので「引き戻し」という名前がついたのです．

5—3—5　整数の加算・減算を行うルーチン

アドレス	＄ＢＤ５２，＄ＢＤ４４
機　能	＄ＢＤ５２：ＦＡＣ１にＦＡＣ２を加える． ＄ＢＤ４４：ＦＡＣ１からＦＡＣ２を引く．
レジスタ	Ａ，Ｂ，Ｕ，Ｘ
入力情報	ＦＡＣ１　　＝被加（減）算数（整数型） ＦＡＣ２　　＝加（減）算数（整数型） （００１７）＝＄０２（整数型）
復帰情報	ＦＡＣ１　　＝結果（整数の範囲を越えた場合は単精度型になる）． （００１７）＝ＦＡＣ１の数値型を示す値
ＷＯＲＫ	ＦＡＣ１，ＦＡＣ２
ＳＵＢ	＄ＢＤ４Ａ→Ｄレジスタを（００７６：００７７）に入れる ＄ＢＤ４Ａ→実数型のときにＸレジスタの示すアドレスに飛ぶ ＄ＡＦ１Ｄ→実数型の加算 ＄ＡＦ１４→実数型の減算 ＄ＢＣＦＦ→整数を単精度実数へ　５－２－５参照 ＄Ｂ３８０→ＦＡＣ１をＦＡＣ２へ　５－２－６参照 ＄ＢＤ０１→整数を単精度実数へ
終了条件	結果が－３２７６８～３２７６７の範囲外のときは，単精度型の演算をする．

解　説　この演算は数値が２の補数表現で代入されているため，符号は考えずにそのまま足し算や引き算を行っています．さらに６８０９では２バイトの演算ＡＤＤＤやＳＵＢＤがありますので，非常に簡単なプログラムで行っています．

５－３－６　整数の乗算を行うルーチン

アドレス　＄ＢＤ７８

機　能　ＦＡＣ１とＦＡＣ２を乗算を行う．ＦＡＣ１とＦＡＣ２は整数でなければならない．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　ＦＡＣ１　　＝整数（被乗算数値）
　ＦＡＣ２　　＝整数（乗算数値）
　（００１７）＝＄０２（整数型）

復帰情報　ＦＡＣ１　　＝結果（整数型あるいは単精度実数）
　（００１７）＝結果が巣精度型の場合＄０４が入る（他は＄０２）

ＷＯＲＫ　ＦＡＣ１，ＦＡＣ２，ＦＡＣ３
　（００Ｅ２）＝掛けるバイトの保存

ＳＵＢ　＄ＢＤ５Ｃ→整数を単精度実数へ変換して演算ルーチンへ
　※ＳＢＣＦＦ→整数を単精度実数へ　５－２－５参照
　※＄Ｂ３８０→ＦＡＣ１をＦＡＣ２へ　５－２－６参照
　以上の※のルーチンは＄ＢＤ５Ｃ内部にあります．
　＄ＢＤＤ９→負の数を正の数へ
　＄ＢＤＢ９→結果が３２７６８になったときの処理
　＄ＢＥ０３→ＦＡＣの符号を反転（整数型）
　＄ＢＤ２９→実数型のときＸレジスタが示す番地へ飛ぶ

終了条件　結果が整数の扱える範囲を越えた場合は実数の計算をする．

解　説　整数型相互の乗算は実数の仮数部の乗算の仕方とほぼ同じです．しかし，整数型の乗算では結果が２バイトを越えてしまう場合があります．このときはＦＡＣ１とＦＡＣ２を単精度実数に変換して実数の乗算を実行します．

　では図５・３・５を簡単に説明します．まずＦＡＣ１の値とＦＡＣ２の値を正の数にします．後は図のようにして１０進数と同様に１バイトを１０進数１桁と考えて掛け合わせていきます．その結果を下に書き，下位からＦＡＣ１につめていきます（ここが実数の仮数部と違います）．

図５・３・５　整数の乗算

		0076	0077	FAC１
×		0084	0085	FAC２
		0077×0085		
	0076×0085			
	0077×0084			
0076×0084				

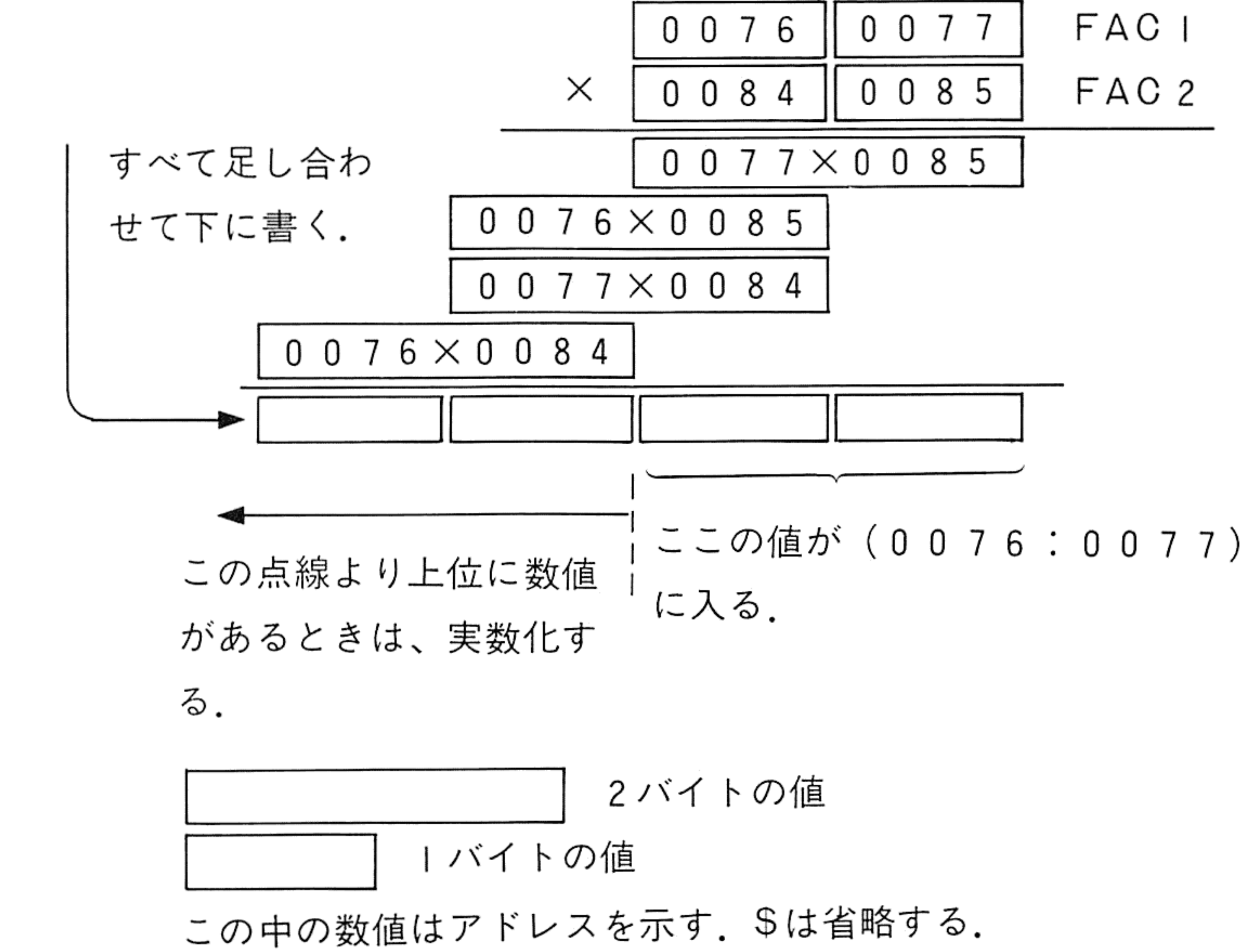
すべて足し合わせて下に書く．

この点線より上位に数値があるときは、実数化する．

ここの値が（００７６：００７７）に入る．

□ ２バイトの値

□ １バイトの値

この中の数値はアドレスを示す．＄は省略する．

〈例〉　FAC１＝＄００A０＝１６０
　　　FAC２＝＄０１０B＝２６７

```
             00   A0        FAC1
             01   0B        FAC2
      ---------------------
             06   E0     (A0)×(0B)
        00   00          (00)×(0B)
        00   A0          (A0)×(01)
   00   00               (00)×(00)
   ------------------------
             A6   E0
```

＄A６E０＝４２７２０となる．

5－3－7　整数の除算・余りを求めるルーチン

アドレス	＄ＢＥ０Ｃ，＄ＢＥ３３
機　能	＄ＢＥ０Ｃ：ＦＡＣ１をＦＡＣ２で割った商を求める．
	＄ＢＥ３３：ＦＡＣ１をＦＡＣ２で割った余りを求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　　＝被除数（整数型）
	ＦＡＣ２　　＝除数　（整数型）
	（００１７）＝＄０２（整数型）
	Ａレジスタ　＝＄００７６番地の値（余りを求めるとき）
復帰情報	ＦＡＣ１　　＝結果（整数型）〔結果が（＋３２７６８）このときは単精度型となる〕
	（００１７）＝結果が単精度型のときは＄０４が入る（他は＄０２）
WORK	ＦＡＣ１，ＦＡＣ２
	（００８Ｂ）＝結果の符号
ＳＵＢ	＄ＢＤＤ９→符号の処理，引数の退避
	＄ＢＥ２２→演算ルーチン
	＄ＢＤＢ２→演算後の処理：整数→実数など
	＄ＢＥ０１→ＦＡＣの符号を反転（整数型）
終了条件	ＦＡＣ２が０のときはError(Division By Zero)を発生する．

解　説　実数の項でも述べましたがＦ－ＢＡＳＩＣでは除算に引き戻し法という方法を用いています．しかし整数型の場合は実数型と異なり完全な引き戻し法を実行しています．では，正の整数についての引き戻し法の方法を示します．５－３－４の方法と比較して下さい．

簡単に，被除数を８ビット，除数を４ビットの正の整数と考えます．図５・３・６を御覧下さい．(4)までは図５・３・４の(6)までとほとんど同じですが図５・３・４では上の数と下の数を比較するのに対し，図５・３・６では何も考えずに上の数から下の数を引いてしまいます．(5)～(6)にかけてが実数のときとの違いですが，このときは引いた数が負になっていますので答えに０を入れ，除数を足し直します（これが引き戻し法の名の由来です）．

次にＦ－ＢＡＳＩＣ上での簡単なアルゴリズムを述べます．

図5・3・6

(1)

```
        ____________
1001 ) 11101111
```

(2)

```
        ____________
1001 ) 11101111
       1001
       ____________
        101
```

(3)

```
         1
        ____________
1001 ) 11101111
       1001
       ____________
        101
```

図5・3・4と違ってここでは上から下を引く．

この値を調べて正なら1，負なら0を上に書く（この場合は正であるから1を書く）．

(4)

```
         11
        ____________
1001 ) 11101111
       1001↓
       ____________
        1011 ┐
        1001 ┘ 引く
       ____________
        0101
```

この場合も正なので1を上に書く．

(5)

```
         11
        ____________
1001 ) 11101111
       1001
       ____________
        1011
        1001
       ____________
       ○0101 ┐
         1001 ┘ 引く
       ____________
         1100
```

ここに1があるつもりで引く

この場合は上から1桁借りてきたことになるので，負の値を示している．

(6)

```
         110
        ____________
1001 ) 11101111
       1001
       ____________
        1011
        1001
       ____________
         0101
         1001
       ____________
         1100 ┐
       + 1001 ┘
       ____________
         0101
```

前の値が負なので0を入れる．

ここの値が負のときは除数を定し直す．

(7) あとは同じ要領で答えを求める．

(8)

```
         11010
        ____________
1001 ) 11101111
       1001
       ____________
        1011
        1001
       ____________
         0101
         1001
       ____________
         1100      ←負の数である．
       + 1001
       ____________
         01011
          1001
       ____________
            101
           1001
       ____________
           1000←負の数である．
       +   1001
       ____________
            101    余り
```

⑴　＄００８Ｂ番地に演算結果の符号を入れます（ＦＡＣ１とＦＡＣ２の符号を比べ異符号ならb_7に１を入れ同符号なら０を入れます）.

⑵　ＦＡＣ１とＦＡＣ２の絶対値をとります.

⑶　引き戻し法により絶対値の商と余りを求めます.

⑷　＄００８Ｂ番地や＄００７６番地の内容によって商と余りの符号を決定します.

なお⑶で－３２７６８／－１を実行するとＦＡＣには単精度実数として答えが入ります.

５—４　　単項演算

以下にあげるルーチンは単項演算（引数が一つの関数）ですが，前部にあるＳＩＮ，ＣＯＳ，ＴＡＮ，ＥＸＰ，ＳＱＲ，ＬＯＧ，ＡＴＮは単精度型の場合しか利用できません．

次にこれらの単項演算の簡単な使い方で，使い方によってはかなり高速に動くプログラムの一例をあげておきます．

〈ＵＳＲ関数の使い方〉

(1)　単項演算とＵＳＲ関数

ＵＳＲ関数は，例１のようなＢＡＳＩＣプログラムを実行した場合，Ｘをマシン語のメモリ上に保存した後＄５０００番地からのマシン語プログラムを実行します（Ｆ—ＢＡＳＩＣ文法書Ｐ－３－４２参照）．

このとき注意していただきたいのは引数が数値型の場合ＦＡＣ１に格納され，＄００１７番地に数値型が代入されるということです．しかもＲＴＳなどで制御がＢＡＳＩＣ上に移るとＦＡＣ１の値がそのままＹに代入されます．ですから例２の囲みの中に，この節であげるサブルーチンのアドレスを代入すればＵＳＲ０（Ｘ）がその関数になるわけです．例３ではＵＳＲ０（Ｘ）＝ＣＯＳ（Ｘ）となり，例４ではＵＳＲ０（Ｘ）＝ＳＩＮ（Ｘ）となります．

〈例１〉

```
１００　ＩＮＰＵＴ　Ｘ
２００　ＤＥＦ　ＵＳＲ０＝＆Ｈ５０００
３００　Ｙ＝ＵＳＲ０（Ｘ）
```

〈例２〉

```
ＪＳＲ　[          ]
ＲＴＳ
```

〈例３〉　ＣＯＳを行う

```
ＪＳＲ　＄ＢＥ６０　 ┐ ＄５０００番地よ
ＲＴＳ　　　　　　　 ┘ り代入
```

〈例４〉　ＳＩＮを行う

```
ＪＳＲ　＄ＢＥ６６　 ┐ ＄５０００番地よ
ＲＴＳ　　　　　　　 ┘ り代入
```

※マシン語はＢＡＳＩＣを動かす前に入れておきます．

ただし，入力情報の数値型が引数と一致していなければなりません．ですから例３でＸの代りにＸ＄などは使えません．

(2) 関数の高速化

(1)の応用になりますが，例５のようなことをＵＳＲ関数で行いたい場合は，例６のプログラムを書き，その後例１のＢＡＳＩＣプログラムを実行すればＹに結果が代入されます．

つまり，ＵＳＲ０（Ｘ）＝ＳＩＮ（ＣＯＳ（ＴＡＮ（Ｘ）））となるわけです．

これらのプログラムは単項演算の計算回数が多いほど，単にＢＡＳＩＣを使うよりも高速になります．

さらに例７のような関数も５－３節や５－２節のサブルーチンを利用して例８のようにできます．ここで注意していただきたいのは数値型で，入力情報や復帰情報を十分理解した上で行って下さい．

〈例５〉

```
SIN(COS(TAN(X)))
```

〈例６〉

```
JSR  $BEB1
JSR  $BE60
JSR  $BE66
RTS
```

〈例７〉

```
SIN(X+COS(X))
```

〈例８〉

```
LDX  $6000
JSR  $B337
JSR  $BE60   :COS
LDX  $6000
LDA  #$00  } 倍精度フラグを
STA  $0015 } $00にする．
JSR  $AF1A
JSR  $BE66   :SIN
RTS
```

＄Ｂ３３７：メモリにＦＡＣ１を保存
＄ＡＦ１Ａ：メモリとＦＡＣ１の加算
５－３－１参照

※　マシン語は＄５０００番地から代入します．

5—4—1　多項式演算を行うルーチン

アドレス	＄ＢＢ９４，＄ＢＢＡ３
機　能	引数の多項式演算を実行する．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	Ｘレジスタ　＝図５・４・１で示したようにデータの入っている箇所の先頭アドレス （００１７）＝単精度型を示す数値（＄０４） ＦＡＣ１　＝単精度実数
復帰情報	ＦＡＣ１　＝多項式演算の結果（単精度型）
WORK	（８Ｄ：８Ｃ）＝Ｘレジスタの保存 （００７Ｄ）　＝データの個数を格納
ＳＵＢ	＄Ｂ３３０→ＦＡＣ１を（００５Ｆ）からの４バイトに格納 ＄Ｂ０ＥＥ→乗算を行うルーチン　５—３—３参照 ＄Ｂ３２Ｂ→ＦＡＣ１を（００６３）からの４バイトに格納 ＄ＡＦ１Ａ→加算を行うルーチン　５—３—１参照

解　説　ＴＡＮ，ＣＯＳ，ＳＩＮ，ＡＴＮ，ＬＯＧ，ＥＸＰという演算を行う際には，必ず近似多項式を用いますが，この近似多項式を計算するのがこのルーチンです．

では具体的な式をあげて説明します．

● ＄ＢＢ９４番地の場合

引数をＸとし結果をＹとすると次の多項式を計算します．

$$Y = X\ (a_0 + a_1X^2 + a_2X^4 + a_3X^6 \cdots\cdots\cdots a_nX^{2n})$$

ここでa_iは図５・４・１のようにあらかじめメモリ上に格納しておかなければなりません．まずＸレジスタの指す番地にｎ（データの数—１）を代入します．その次のバイトから４バイトずつ，定数をa_nから順に変数テーブル用の単精度実数の格納法（５－２－６参照）にしたがって格納していきます．

このルーチンのフローチャートを図５・４・２に示しました．このフローチャートは次の式の(　)の一番内部から順に計算しています．

図5・4・1　定数の格納法

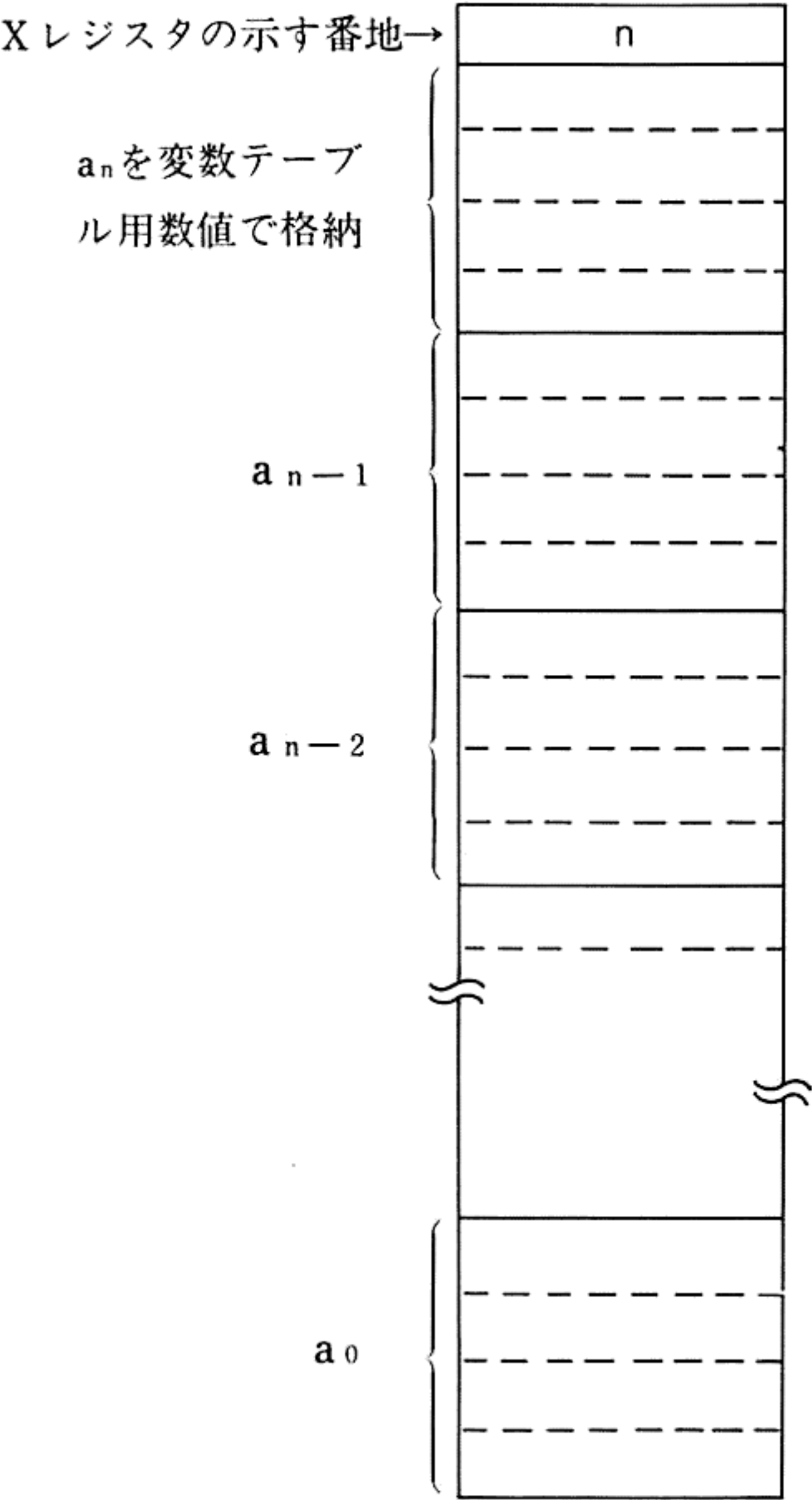



図5・4・2　多項式フローチャート(1)

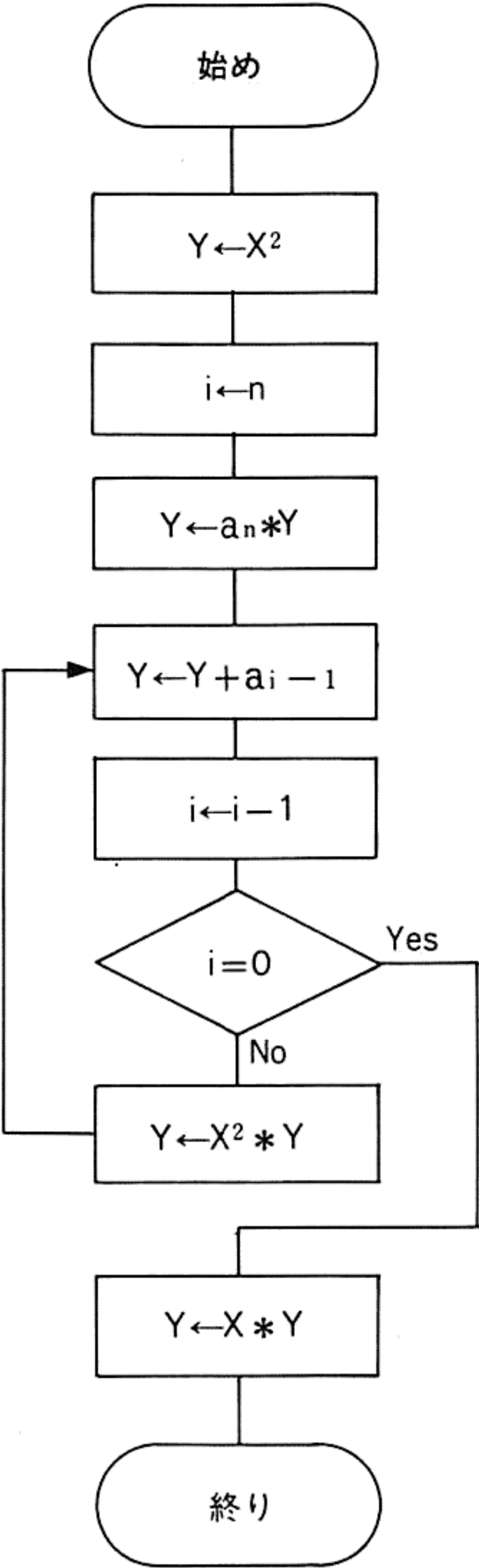
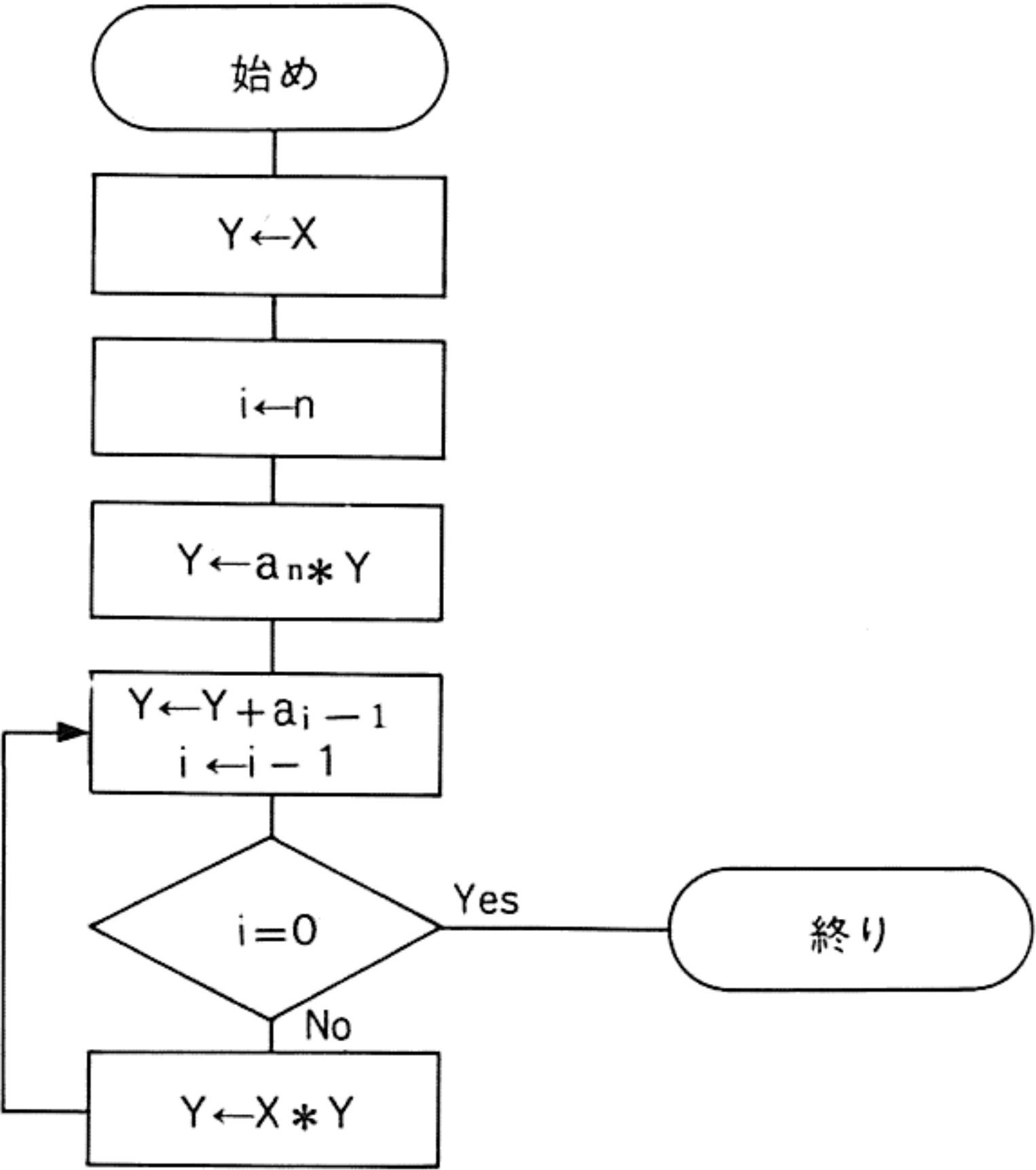



図5・4・3　多項式フローチャート(2)

$$Y=X\ (a_0+X^2\ (a_1+X^2\ (a_2+X^2\ (\cdots\cdots\ (a_{n-1}+a_nX^2))\cdots\ )$$

● ＄ＢＢＡ３番地の場合

引数をＸとし結果をＹとすると次の多項式を計算します.

$$Y=a_0+a_1X+a_2X^2+a_3X^3+a_4X^4\cdots\cdots a_nX^n$$

定数a_iの格納の仕方は＄ＢＢ９４番地の場合と同様です.

このルーチンのフローチャートを**図５・４・３**に示しました. このフロチャートも以下のような計算の一番内側の(　)の内部から演算しています.

$$Y=a_0+X\ (a_1+X\ (a_2+X)\ \cdots\cdots\cdots\ (a_{n-1}+a_nX))))\cdots)$$

サンプル 以下のサンプルは配列を用い, 下の式の定数を代入してＹの値を求めるプログラムです.

$$Y=a_0+a_1X+a_2X^2+a_3X^3+a_4X^4\cdots\cdots a_nX^n$$

配列の格納の仕方は２－２－２を御覧いただければわかりますが**図５・４・１**とほぼ同様です. 異なっているのは**図５・４・１**の先頭の箇所にｎが入っているのに対して配列ではｎ＋１が入り, a_nは配列Ａ(０)の箇所に入るという点です. ですからa_nの値をＡ(０)に代入するのと同様にa_n～a_0を代入します（１７０行目から).

このルーチンへの入力情報はＸレジスタに**図５・４・１**の先頭アドレスですから, ＡＤ％－１をメモリ上の（４ＦＦＥ：４ＦＦＦ）に入れます〔Ａ(０）の先頭アドレス－１に配列の個数が入っている, ２２０行目から〕.

２８０行目より, マシン語 ＄５０００番地からの命令を実行します. ＄５０００番地からのＬＤＸでＸレジスタにデータの先頭アドレスを読み, ＤＥＣでｎ＋１をｎにしておきます. ＵＳＲ０を実行したとき, ＦＡＣ１にＸの内容が入りますので入力情報はこれですべてそろいます. この後＄ＢＢＡ３番地を呼び出せばよいわけです.

```
100 '*********************************************************
110 '****           ﾀｺｳｼｷ ﾉ ｹｲｻﾝ             ****************
120 '*********************************************************
130 X=0:Y=0
140 DEF USR0=&H5000
150 INPUT"N=",N
160 DIM A(N)
170 'ﾃｲｽｳ ﾉ ﾀﾞｲﾆｭｳ
180 FOR I=0 TO N
190 PRINT "A(";N-I;")=";:INPUT A(I)
200 NEXT I
210 INPUT"X=",X
220 AD%=VARPTR(A(0))
230 AD%=AD%-1
240 'ﾊｲﾚﾂ ﾉ Address ｦ ﾒﾓﾘ ﾍ
250 POKE &H4FFE,AD% ¥ 256
260 POKE &H4FFF,AD% MOD 256
270 'ﾀｺｳｼｷ ﾉ ｼﾞｯｺｳ
280 Y=USR0(X)
290 PRINT"Y=";Y
300 END
```

```
PAGE  001 (821201,033302) SAMPLE

00010                           * ﾀｺｳｼｷ
00020                                  NAM   SAMPLE
00030  5000                            ORG   $5000
00040                                  OPT   MEM
00050  5000 BE   4FFE                  LDX   $4FFE     ﾊｲﾚﾂ ﾉ Address
00060  5003 6A   84                    DEC   ,X        (ﾊｲﾚﾂ ﾉ ｺｽｳ)-1
00070  5005 BD   BBA3                  JSR   $BBA3     ﾀｺｳｼｷ ﾉ ｹｲｻﾝ
00080  5008 6C   9F 4FFE               INC   [$4FFE]
00090  500C 39                         RTS
00100                                  END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000
```

```
RUN
N=5
A( 5 )=? 2
A( 4 )=? 2
A( 3 )=? 4
A( 2 )=? 3
A( 1 )=? 5
A( 0 )=? 2
X=2
Y= 152

Ready
HARDC
```

5―4―2　SIN・COSを求めるルーチン

アドレス	$BE60，$BE66
機　　能	$BE60　引数（単精度実数）のコサインを求める．
	$BE66　引数（単精度実数）のサインを求める．
レジスタ	A，B，X，U
入力情報	FAC1　　＝単精度実数の引数
	（0017）＝単精度型を示す数値（$04）
復帰情報	FAC1　　＝SIN，COSの結果（単精度実数）
WORK	FAC1，FAC2
	（008B）
SUB	$B380→FAC1をFAC2へ転送する　5－2－6参照
	$B22D→FAC2を変数テーブル用数値で割る
	$B472→FAC1のINTをとる　5－4－8参照
	$AF14→FAC1をFAC2から引く
	$AF11→減算ルーチン
	$BB2F→符号反転
	$AF1A→加算ルーチン
	$BB94→近似多項式　5－4－1参照

解　　説　以下にSIN，COSのアルゴリズムを述べます．

●SINについて

ここでは引数（FAC1に代入する値）をXとします．

(1)　Xが非常に小さいとき（絶対値が9．76562×10^{-4}以下のとき）

次の公式を使います．他のときは(2)へいきます．

$$\mathrm{SIN}(X)=X \quad (X\fallingdotseq 0)$$

(2)　X＝Y＋2nπと置きます（nは整数）．ここで次の公式を利用します．

$$\mathrm{SIN}(X)=\mathrm{SIN}(Y+2n\pi)=\mathrm{SIN}(Y) \quad (0\leqq Y<2\pi)$$

変形して

$$\mathrm{SIN}\{2\pi(\frac{Y}{2\pi}+n)\}=\mathrm{SIN}(Y) \cdots\cdots①$$

そこで①式の左辺の(　)の中のＩＮＴをとるとｎになるのに注目して$\frac{Y}{2\pi}$を計算します.

$$\frac{Y}{2\pi}=\frac{X}{2\pi}-INT\left(\frac{X}{2\pi}\right)$$

このＹの値によって場合分けをします. **図５・４・４**を御覧下さい.

(i)　$0 \leqq Y < \pi/2$　　図ではⅠの領域

Ｙはそのままで(3)を実行します.

(ii)　$\pi/2 \leqq Y < 3\pi/2$　　図ではⅡ，Ⅲの領域

ＳＩＮ（Ｙ）＝ＳＩＮ（π－Ｙ）

という公式を利用しますので

Ｙ←π－Ｙ　　と置き直します.

(3)へ飛びます.

(iii)　$3\pi/2 \leqq Y < 2\pi$　図ではⅣの領域

ＳＩＮ（Ｙ）＝ＳＩＮ（Ｙ－２π）

という公式より

Ｙ←Ｙ－２π　と置き直します.

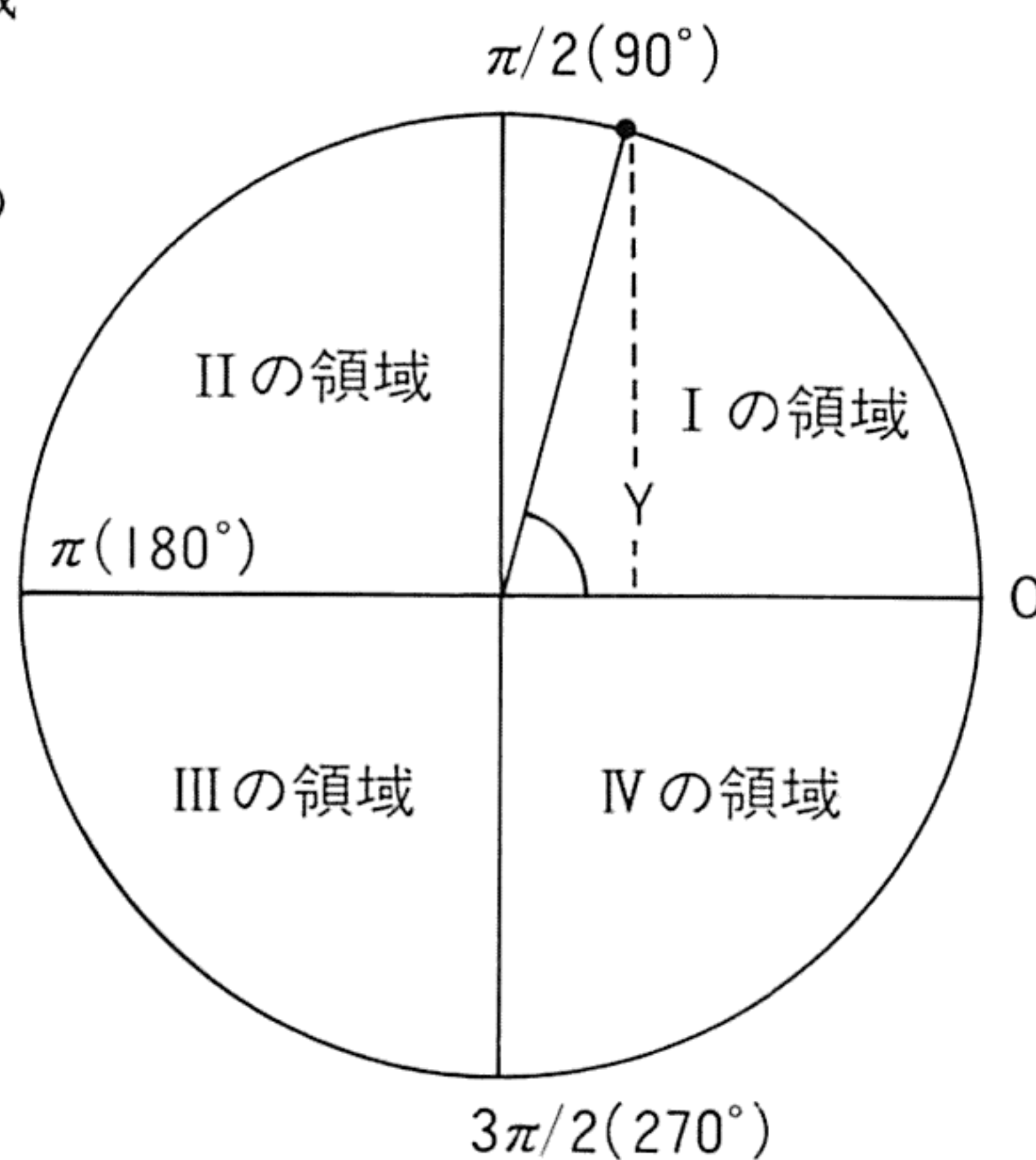
(3)へ飛びます.

※　(i)，(ii)，(iii)を計算することで，(3)の近似計算の適応できる範囲のＹの値にＹを変換します.

図５・４・４　単位円

(3)　次の近似計算を実行します.

$$SIN(Y) \fallingdotseq \frac{Y}{2\pi}\left\{a_0+a_1\left(\frac{Y}{2\pi}\right)^2+a_2\left(\frac{Y}{2\pi}\right)^4+a_3\left(\frac{Y}{2\pi}\right)^6+a_4\left(\frac{Y}{2\pi}\right)^8\right\}$$

$a_0=6.2831850$

$a_1=-41.341675$

$a_2=81.602234$

$a_3=-76.57498$

$a_4=39.710670$

この結果がＳＩＮ（Ｘ）の結果となります．

●ＣＯＳについて

ＣＯＳは以下の公式より計算します．

$$\mathrm{COS}(X) = \mathrm{SIN}\left(\frac{\pi}{2} + X\right)$$

ですからＣＯＳを求めるには Ｘにπ／２を足してからＳＩＮを求めればよいのです．

〈ＳＩＮ，ＣＯＳのＢＡＳＩＣによるシミュレーション〉

以上のことをＢＡＳＩＣ上で行ってみました．定数などは文章中のものより桁数を多く求めてあります．ここで用いたＹ＃は文中のＹ／２πに相当します．

```
100 '********** BASIC SIN SIMULATION ********************
110 '*************************************************
120 ' 3.141592654;ﾊﾟｲ
130 '*************************************************
140 INPUT "X=";X#:Z#=X#
150 'X ｶﾞ 0 ﾆ ﾁｶｲﾄｷ ﾉ ｼｮﾘ
160 IF ABS(X#)<.00097652# THEN G#=X#:GOTO 230
170 Y#=(X#/3.141592654#/2-INT(X#/3.141592654#/2))
180 'ﾍﾝｽｳ ﾊﾝｲ ﾆﾖﾙ ﾊﾞｱｲﾜｹ
190 IF Y#<.25 GOTO 220
200 IF Y#<.75 THEN Y#=.5-Y# ELSE Y#=Y#-1
210 'ｷﾝｼﾞﾀｺｳ ｼｷ ﾉ ｹｲｻﾝ
220 G#=Y#*(6.283185#+-41.341675#*Y#*Y#+81.602234#*Y#*Y#*Y#*Y#+-76.57498#*Y#*Y#*Y
#*Y#*Y#*Y#+39.71067#*Y#*Y#*Y#*Y#*Y#*Y#*Y#*Y#)
230 PRINT "BASIC SIN(";Z#;")=";G#,"SIN(";Z#;")="; SIN(Z#)
240 END
```

```
100 '********** BASIC COS SIMULATION ********************
110 '*************************************************
120 ' 3.141592654;ﾊﾟｲ
130 '*************************************************
140 INPUT "X=";X#:Z#=X#
150 X#=3.141592654#/2+X#
160 'X ｶﾞ 0 ﾆ ﾁｶｲﾄｷ ﾉ ｼｮﾘ
170 IF ABS(X#)<.00097652# THEN G#=X#:GOTO 240
180 Y#=(X#/3.141592654#/2-INT(X#/3.141592654#/2))
190 'ﾍﾝｽｳ ﾊﾝｲ ﾆﾖﾙ ﾊﾞｱｲﾜｹ
200 IF Y#<.25 GOTO 230
210 IF Y#<.75 THEN Y#=.5-Y# ELSE Y#=Y#-1
220 'ｷﾝｼﾞﾀｺｳ ｼｷ ﾉ ｹｲｻﾝ
230 G#=Y#*(6.283185#+-41.341675#*Y#*Y#+81.602234#*Y#*Y#*Y#*Y#+-76.57498#*Y#*Y#*Y
#*Y#*Y#*Y#+39.71067#*Y#*Y#*Y#*Y#*Y#*Y#*Y#*Y#)
240 PRINT "BASIC COS(";Z#;")=";G#,"COS(";Z#;")="; COS(Z#)
250 END
```

5－4－3　TANを求めるルーチン

アドレス	＄BEB1
機　　能	引数（単精度実数）のTANを求める．
レジスタ	A，B，X，U
入力情報	FAC1　　＝単精度実数の引数 （0017）＝単精度実数を示す数値（＄04）
復帰情報	FAC1　　＝TANの結果（単精度実数）
WORK	（5F～62）＝FAC1の保存 （6B～6E）＝FAC1の保存 （007C）　＝FAC1の符号部 （001C）　＝TAN場合分けフラグ
SUB	＄B330→FAC1を（005F）からの4バイトに保存 ＄BE66→SIN　5－4－2参照 ＄B337→FAC1をメモリ上へ変数テーブル用数値で保存 ＄BED0→SINの途中へ ＄B234→除算ルーチン　5－2－4参照 ＄B301→メモリ上からFAC1へ数値を代入
終了条件	1.57079632～1.57079643の数値とこれにnπを加えたものはDivision By Zerōを発生する．

解　　説　TANは基本的には以下の式を実行しています．

$$\mathrm{TAN}\ (X) = \frac{\mathrm{SIN}\ (X)}{\mathrm{COS}\ (X)}$$

しかし，SUBの項を御覧いただければわかりますが，COSのアドレスを呼び出していません．これはSINを求めた後にまだメモリ上にY／2πの値が保存されているためです．この際にπ／2≦Y＜3π／2のとき＄001C番地にフラグを立てておきます．TANが実行されSINのルーチンが終ると，このフラグを調べます．このフラグが立っている場合はY／2πから1／4を引いて，立っていないときは，1／4からY／2πを引きます．この値を近似多項式に代入するとCOSが求まるわけです．

5—4—4　ATNを求めるルーチン

アドレス	$BEF5
機　　能	引数（単精度実数）のATNを求める．
レジスタ	A，B，X，U
入力情報	FAC1　　＝単精度実数の引数 （0017）＝単精度型を示す数値（$04）
復帰情報	FAC1　　＝ATNの結果（単精度実数）
WORK	（007C）
SUB	$BF1F→$BB2F：符号反転ルーチン $BECD→$B234：除算を行う　5—3—4参照 $BEAE→$BB94：近似多項式　5—4—1参照 $AF11→減算ルーチン　5—3—2参照

解　　説　アルゴリズムは以下のとおりです（引数をXとします）．

(1)　まず次の4つの場合に分けそれぞれ処理を行います．

(i)　$0 \leqq X < 1$のとき

Xの値はそのままで(2)を実行します．

(ii)　$1 \leqq X$のとき以下の公式を利用します．

$$\mathrm{ATN}(X) = \frac{\pi}{2} - \mathrm{ATN}\left(\frac{1}{X}\right)$$

X←1／X　と置いて(2)の近似式を行った後，π／2から結果を引いて答えを出します．

(iii)　$-1 < X < 0$のとき

以下の公式を利用します．

$$\mathrm{ATN}(-X) = -\mathrm{ATN}(X)$$

X←－Xと置いて(2)の近似式を実行した後，符号を反転させます．

(iv)　$X \leqq -1$

この場合は(ii)の公式と(iii)の公式を利用します．つまり次の公式を実行して計算するわけです．

$$\mathrm{ATN}(-X) = -\frac{\pi}{2} + \mathrm{ATN}\left(\frac{1}{X}\right)$$

X←－1／Xと置いて(2)の近似式を行った後，結果からπ／2を引いて答えを求めます．

(2) 次の近似式を行います.

$$\mathrm{ATN}(X) \fallingdotseq X(a_0 + a_1X^2 + a_2X^4 + a_3X^6 + a_4X^8 + a_5X^{10} + a_6X^{12} + a_7X^{14} + a_8X^{16})$$

$a_8 = 2.86623 \times 10^{-3}$

$a_7 = -1.61657 \times 10^{-2}$

$a_6 = 4.29096 \times 10^{-2}$

$a_5 = -7.52896 \times 10^{-2}$

$a_4 = 0.106563$

$a_3 = -0.142089$

$a_2 = 0.199936$

$a_1 = -0.33331$

$a_0 = 1.00000$

(3) 上の近似式を実行したのち，(1)の場合分けにしたがってＡＴＮ（Ｘ）を求めます.

※ (1)の場合分けについて

(1)の場合分けを実行することにより，すべてのＸの値のＡＴＮ（Ｘ）が０≦Ｘ＜１のときのＡＴＮ(Ｘ)を，求める近似式より得ることができるわけです.

〈ＡＴＮのＢＡＳＩＣによるシミュレーション〉

１４０行目と１７０・１８０行目は以上で述べた場合分けを実行しています.この計算では倍精度実数を使っていないため，１の付近でかなりの誤差を生じてしまいます.

```
100 '*****************************************************************
110 '********             BASIC ATN SIMULATION            ******************
120 '*****************************************************************
130 INPUT"X=";X
140 IF 1<=ABS(X) THEN Z=1/ABS(X) ELSE Z=ABS(X)
150 '*******      KINJI TAKOUSHIKI          *********************************
160 COTAE=Z*(1+Z*Z*(-.333315+Z*Z*(.199936+Z*Z*(-.142089+Z*Z*(.106563+Z*Z*(-7.528
96E-02+Z*Z*(4.29096E-02+Z*Z*(-1.61657E-02+Z*Z*2.86623E-03))))))))
170 IF 1<=ABS(X) THEN COTAE=3.14159/2-COTAE
180 COTAE=SGN(X)*COTAE
190 PRINT"BASIC ATN(";X;")=";COTAE,"ATN(";X;")=";ATN(X)
200 END
```

5—4—5　ＥＸＰを求めるルーチン

アドレス	＄ＢＢ５Ｂ
機　能	引数（単精度実数）のＥＸＰを求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　　＝単精度実数の引数 （００１７）＝単精度を示す数値（＄０４）
復帰情報	ＦＡＣ１　　＝ＴＡＮの結果（単精度実数）
ＷＯＲＫ	（００１１），（００８Ｂ）
ＳＵＢ	＄Ｂ３３０→ＦＡＣ１を（００５Ｆ）からの４バイトに保存 ＄ＢＢＡ０→＄Ｂ０ＥＥ：乗算ルーチンへ　５—３—３参照 ＄Ｂ１Ｆ５→Error 処理 ＄Ｂ４７２→ＩＮＴ　５—４—８参照 ＄ＡＦ１Ａ→加算ルーチン　５—３—１参照 ＄Ｂ０ＥＢ→log２＊ＦＡＣ１ ＄ＢＢ２Ｆ→符号反転ルーチン ＄ＢＢＡ３→近似多項式ルーチン　５—４—１参照 ＄ＡＦ１１→減算ルーチン　５—３—２参照
終了条件	結果が 1.7014×10^{38} を越えた場合はError(Overflow)を発生する．

解　説　ＥＸＰ（Ｘ）の計算の一般的な方法は以下のとおりです．

$$EXP(X)=EXP(n\log 2+Y)=2^{n}*EXP(Y)$$

このＥＸＰ（Ｙ）とｎを求めればＥＸＰ（Ｘ）を求めることができます（ｎは整数，$0\leqq Y<\log 2$）．次にＦ－ＢＡＳＩＣ上でのアルゴリズムを述べていきます．

(1)　Ｘをlog２で割ります．Ｚ＝Ｘ／log２と置きます．
　〔このとき，ＩＮＴ（Ｚ）がｎに相当します〕

(2)　Ｚの値によって以下のように場合分けします．
　(i)　Ｚの値が１２８以上のとき（Ｘが８８．７２２９以上のとき）Overflowを発生します．
　(ii)　Ｚの値が－１２８以下のとき（Ｘが－８８．７２２９以下のとき）ＦＡＣ１に０を代入して終ります．
　(iii)　それ以外は(3)を実行します．

(3)　Ｚの値のＩＮＴを取ります．この値が１２７のとき（Ｘが８８．０３のとき）はOverflowを発生します．それ以外のときは(4)へ行きます．

(4)　ＥＸＰ（Ｙ）を求めます．ＥＸＰ（Ｙ）は近似計算で求めますが，まず以下のような準備をします．

$$U = X - \log 2 * (INT(Z) + 1) = Y - \log 2$$

$V = -U$　と置きます．

(5)　以下の近似計算をしますがこの近似計算は｛ＥＸＰ（Ｙ）｝／２を求めることになります．

$$EXP(Y)/2 \fallingdotseq a_0 + a_1 V + a_2 V^2 + a_3 V^3 + a_4 V^4 + a_5 V^5 + a_6 V^6 + a_7 V^7$$

$a_0 = 1.0$

$a_1 = -1.0$

$a_2 = 0.5$

$a_3 = -0.166665$

$a_4 = 4.16574 \times 10^{-2}$

$a_5 = -8.30136 \times 10^{-3}$

$a_6 = 1.32988 \times 10^{-3}$

$a_7 = -1.41316 \times 10^{-4}$

(6)　$EXP(X) = 2^{n+1} * EXP(Y)/2$ より求めます．具体的にはｎ＋１をＥＸＰ（Ｙ）／２の指数部に足します．

〈ＥＸＰのＢＡＳＩＣによるシミュレーション〉

```
100 '**************************************************************
110 '********************     BASIC EXP SIMULATION    ********************
120 '**************************************************************
130 '        1.44270     ;1/LOG(2)
140 '**************************************************************
150 INPUT"X=";X
160 Z=X*1.4427
170 'Error ｼｮﾘﾄ ｺﾀｴ ｶﾞ 0 ﾆ ﾁｶｲﾄｷ ﾉ ｼｮﾘ
180 IF Z=>128 THEN PRINT "Overflow":BEEP:END
190 IF Z<-128 THEN COTAE=0:GOTO290
200 IF INT(Z)=127THEN PRINT "Overflow":BEEP:END
210 U=X-.693147*(INT(Z)+1)
220 V=-U
230 'ｷﾝｼﾞﾀｺｳ ｼｷ ﾉ ｹｲｻﾝ
240 COTAE=1!+V*(-1!+V*(.5+V*(-.166665+V*(4.16574E-02+V*(-8.30101E-03+V*(1.32988E
-03+V*-1.41316E-04)))))))
250 '2 ﾉ n+1 ｼﾞｮｳ ｦ ｶｹﾙ
260 FOR I=0 TO INT(Z)
270 COTAE=2*COTAE
280 NEXT I
290 PRINT "BASIC EXP(";X;")=";COTAE,"EXP(";X;")=";EXP(X)
300 END
```

5—4—6　ＬＯＧを求めるルーチン

アドレス	＄Ｂ０ＡＡ
機　能	引数（単精度実数）のＬＯＧを求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　　＝単精度実数の引数
	（００１７）＝単精度型を示す数値（＄０４）
復帰情報	ＦＡＣ１　　＝ＬＯＧの結果を単精度実数で代入
ＷＯＲＫ	（００７４）＝ＦＡＣ１の指数部
ＳＵＢ	＄Ｂ３Ａ０→符号判定ルーチン
	＄Ｂ０Ｂ２→符号反転ルーチン
	＄９６６３→Illegal Function Call
	＄ＡＦ１Ａ→加算ルーチン　５—３—１参照
	＄Ｂ２３４→除算ルーチン　５—３—４参照
	＄ＡＦ１１→減算ルーチン　５—３—２参照
	＄ＢＢ９４→近似多項式ルーチン　５—４—１参照
	＄Ｂ３３０→（００５Ｆ）からの数値データをＦＡＣ１へ
	＄Ｂ３Ｃ０→Ｂレジスタの値を整数型にする
	＄ＢＣＦＦ→整数型を単精度型へ　５—２—５参照
	※＄Ｂ０ＥＥ→乗算ルーチン　５—３—３参照
	※このルーチンはＬＯＧの内部にある．
終了条件	引数が負の数のときはIllegal Function Callを発生する．

解　説　一般にコンピュータでＬＯＧを求めるためには内部数値型式が２進数であることを利用して以下の公式を利用します．

$$\mathrm{LOG}(X)=\mathrm{LOG}(2^{n}*Y)=\log 2*(n+\log_2 Y)$$

ここで　$Z=\dfrac{(Y-\sqrt{2}/2)}{(Y+\sqrt{2}/2)}$ とおきます．

$$f(Z) \fallingdotseq a_0 + a_1 Z^2 + a_2 Z^4$$

$$a_0 = 2.88539$$

$$a_1 = 0.961471$$

$$a_2 = 0.598979$$

ここはF—BASICのものを用いています.

$$LOG(X) = \log 2 * \left\{ n - \frac{1}{2} + Z * f(Z) \right\}$$

$$\log 2 = 0.693147$$

次にF—BASICではどのように行っているかを述べます.

(1) 引数が負あるいは0のときはIllegal Function Callを発生します.

(2) Aレジスタ←(0074)−$80　これがnを求めたことになります.

(3) (0074)←$80

(4) $Z = \dfrac{(Y - \sqrt{2}/2)}{(Y + \sqrt{2}/2)}$ を求めます.

(5) $LOG(X) \fallingdotseq \log 2 * \left\{ n - \frac{1}{2} + Z * f(Z) \right\}$ を計算します.

f(Z)は上のf(Z)と同じです.

〈LOGのBASICによるシミュレーション〉

文中のnがこのプログラム上でIとなっている他は文中の文字を使っています.

```
100 '*************************************************************
110 '*************************************************************
120 '********** BASIC LOG SIMULATION ******************************
130 '*************************************************************
140 '1.41421356;ルート2
150 '0.69314718  ;ログ゛2
160 '*************************************************************
170 INPUT"X=";X#
180 Y#=X#
190 'Error ショリ
200 IF Y#<=0 THEN PRINT "Illegal Function Call":BEEP:END
210 'n ヲ  モトメル
220 I=0
230 Y#=Y#/2
240 I=I+1
250 IF Y#>1 GOTO 230
260 Z#=(Y#-1.41421356#/2)/(Y#+1.41421356#/2)
270 'キンジ゛タコウ シキノ ケイサン
280 G#=.69314718#*(I-1/2+Z#*(2.88539242744#+Z#*Z#*.961470663547516#+Z#*Z#*Z#*Z#*
.598978638648986#))
290 PRINT "BASIC LOG(";X#;")=";G#,"LOG(";X#;")=";LOG(X#)
300 END
```

5−4−7 SQR・べき乗を求めるルーチン

アドレス	$BAEE
機　能	引数（単精度実数）のSQRを求める．
レジスタ	A，B，X，U
入力情報	FAC1　＝単精度実数の引数 （0017）＝単精度実数を示す値（$04）
復帰情報	FAC1　＝引数のSQR（単精度実数）
WORK	（0011） （006B） （008A）＝FAC2の指数部
SUB	$B380→FAC1をFAC2へ　5―2―6参照 $B301→メモリ上の数値をFAC1へ　5―2―6参照 $AFC3→FAC1の数値が0のときFAC1を0とする $B337→FAC1をメモリ上へ変数テーブル用数値で保存 $B472→FAC1のINTをとる　5―4―8参照 $B3F6→FAC1とメモリの値の比較 $B373→FAC2をFAC1へ $B0AA→FAC1のLOGをとる　5―4―6参照 $B0EE→乗算ルーチン　5―3―3参照 $BB5B→FAC1のEXPをとる　5―4―5参照

解　説　SQRは以下の公式を実行します．

$$SQR(X)=EXP\left(\frac{LOG(X)}{2}\right)$$

しかし，このルーチンはただSQR（X）を求めるだけでなく，上の式の1／2をYに変えた場合は次の公式を実行していることになります．

$$X \wedge Y=EXP\{LOG(X)*Y\}$$

ここではこの式のアルゴリズムを述べていきます．SQR（X）はこの式のYの値が1／2のときを考えて下さい．

(1) X＝0のときX＾Y＝0とします．
Y＝0のときX＾Y＝1とします．ただしX＝0，Y＝0のときはX＾Y＝1となります．

(2) Xの符号によって次のように演算の仕方を変えます．

(i) Xが正のとき，そのまま(3)へいきます．

(ii) Xが負のとき
次の演算を実行してその結果によりさらに場合分けします．

Z＝INT（Y）

① Z≒Yのとき（Yが整数でないとき）
Errorを発生します（Illegal Function Call）．

② Z＝Yのとき（Yが整数のとき）
Xの符号を正にして，(3)へいきます．

(3) X＾Y＝EXP｛LOG(X)＊Y｝を実行します．
(ii)の②のときだけ次の操作をします．
Yが偶数のときはそのまま，Yが奇数のときはX＾Yの符号を負にします．
この操作が何を意味するかは例を御覧下さい．

〈例〉 1．(－2）＾2＝2＾2＝4
2．(－2）＾3＝－（2＾3）＝－8
1ではYが偶数ですから　Xを－Xにしても答えは変わりません．
2ではXを－Xにしてから計算した後，答えを負にしなければなりません．

X＾Yでこのルーチンを使うときのアドレスや入力情報をあげておきます．

アドレス	＄BAF7
入力情報	FAC1　＝Yを単精度型で
	FAC2　＝Xを単精度型で
	（0017）＝単精度実数を示す数値（＄04）
	Aレジスタ　＝（0082）
	Bレジスタ　＝（0074）
使用法	必ず
	LDB　＄0074
	JSR　＄BAF7　の順で使用して下さい．

〈SQRのBASICによるシミュレーション〉

```
100 '*******************************************************************
110 '******  BASIC SQR SIMULATION         *******************************
120 '*******************************************************************
130 INPUT"X=";X#
140 IF X#=0 GOTO 180
150 GOSUB 230
160 G#=G#*1/2
170 GOSUB 400
180 PRINT "BASIC SQR(";X#;")=";COTAE,"SQR(";X#;")=";SQR(X#)
190 END
200 '**************** EXP SUB ************************************
210 '1.41421356 ;ﾙｰﾄ2  , 0.69314718  ;ﾛｸﾞ2
220 '*******************************************************************
230 Y#=X#
240 'Error ｼｮﾘ
250 IF Y#<=0 THEN PRINT "Illegal Function Call":BEEP:END
260 'n ｦ  ﾓﾄﾒﾙ
270 I=0
280 Y#=Y#/2
290 I=I+1
300 IF Y#>1 GOTO 280
310 Z#=(Y#-1.41421356#/2)/(Y#+1.41421356#/2)
320 'ｷﾝｼﾞﾀｺｳ ｼｷﾉ ｹｲｻﾝ
330 G#=.69314718#*(I-1/2+Z#*(2.88539242744#+Z#*Z#*.961470663547516#+Z#*Z#*Z#*Z#*
.598978638648986#))
350 RETURN
360 '
370 '*******************    EXP SUB   **************************************
380 '        1.44270     ;1/LOG(2)
390 '*********************************************************************
400 Z=G#*1.4427
410 IF Z=>128 THEN PRINT "Overflow":BEEP:END
420 IF Z<-128 THEN COTAE=0:GOTO500
430 IF INT(Z)=127THEN PRINT "Overflow":BEEP:END
440 U=G#-.693147*(INT(Z)+1)
450 V=-U
460 COTAE=1!+V*(-1!+V*(.5+V*(-.166665+V*(4.16574E-02+V*(-8.30101E-03+V*(1.32988E
-03+V*-1.41316E-04))))))
470 FOR I=0 TO INT(Z)
480 COTAE=2*COTAE
490 NEXT I
500 RETURN
```

5—4—8　ＩＮＴを求めるルーチン

アドレス	＄Ｂ４７２
機　　能	引数の値を越えない整数を求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　　＝引数 （００１７）＝引数の数値の型を示す数値
復帰情報	ＦＡＣ１　　＝引数のＩＮＴを行った結果　数値型は演算前と変わらない （００１１）＝引数が－２５６～２５５の値であればその絶対値が付号なし２進数で保存されている
ＷＯＲＫ	ＦＡＣ１ （００１５）＝倍精度フラグ （００１１）＝復帰情報参照
ＳＵＢ	＄ＢＣＡＥ→数値型判断ルーチン　５－２－１参照 ＄Ｂ４３５→小数点以下の切り捨て ＄ＡＦ９３→正規化ルーチン

解　　説　次のようなアルゴリズムを実行します．

(1)　指数部が単精度型のときは＄９８，倍精度型のときでかつ＄Ｂ８以上のときはなにもせずにリターンします．他の場合は(2)へいきます．

※　ここで行っているのは2^0 の位の位置を調べているのです．指数部が上の数値以上のときはＦＡＣの内部に小数点以下が表現されていないため，ＦＡＣの内容がそのままＩＮＴになるのです．例えば0.10×10^2はＩＮＴをとっても0.10×10^2です．

(2)　ＦＡＣ１（引数）の正負によって場合分けします．

(i)　正の数のとき

そのまま(3)へいきます．

(ii)　負の数のとき

仮数部のＮＥＧ（２の補数）をとります．＄００８１番地に＄ＦＦを代入します．それから(3)へいきます．

(3)　2^0 の位より小さい位（小数点以下）を切り捨てます．

(4)　次のように場合分けします．

(i)　正の数のとき

そのまま(5)へいきます．

(ii)　負の数のとき

仮数部のＮＥＧ（２の補数）をとり，それから(5)へいきます．

(5)　正規化を実行します．

※(2)や(4)を実行する理由

ＩＮＴとは引数Ｘを越えない最大の整数ということです．ですから，ただ仮数部の切り捨てを行ったのではＦＩＸになってしまいますので，このような処理を実行しているわけです．例を使って説明します．

例１を御覧下さい．正の数のＩＮＴを行っています．簡単にするため仮数部を８ビットとします（最初の０と小数点は含まない）．

〈例１〉　Ｘ＝４．７５とします．２進数のＩＮＴは以下のように行います．

切り捨て

$X = 0.10011000 \times 2^3 \longrightarrow 0.00000100 \times 2^8$

正規化

$\longrightarrow \quad X = 0.10000000 \times 2^3$　　これはＸ＝４．００となります．

負の数のときは例２のようになります．

〈例２〉　Ｘ＝－４．７５とします．

仮数部のＮＥＧをとる．

$X = -0.10011000 \times 2^3 \longrightarrow X = -0.01101000 \times 2^3$

切り捨て(左から１を入れていく) 仮数部のＮＥＧをとる．

$\longrightarrow X = -0.11111011 \times 2^8 \longrightarrow X = -0.00000101 \times 2^8$

正規化

$\longrightarrow X = -0.10100000 \times 2^3$　　Ｘ＝－５．０となります．

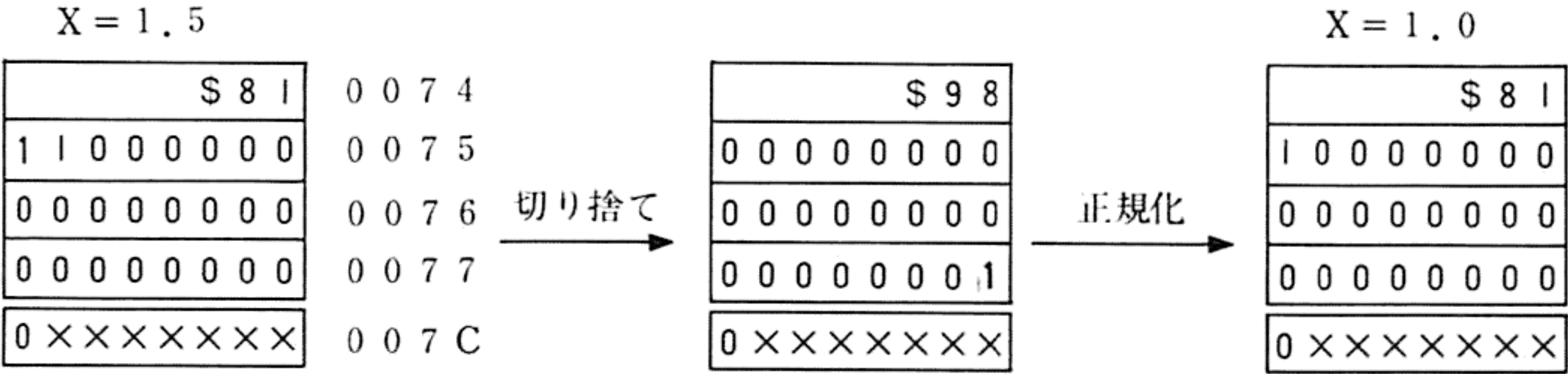
例2で最初の方のＮＥＧを取るのを(2)で行っています．次の箇所が負の数でＩＮＴを求めるときのポイントです．切り捨てを行うには2^0の位が仮数部の右端に来るまでシフトすればよいのですが，負の数の場合には1ビット右へシフトするごとに1を左端から1つ入れていきます〔これが＄0081番地に＄ＦＦを入れる理由(※を見て下さい)〕．この後，仮数部のＮＥＧをとります〔(4)のところ〕．それから正規化すればＸ＝5．0が得られるわけです．

※　左から1を入れる理由

負の数の場合1を切り捨てるのではなく，0を切り捨てることになります．つまり0と1の立場が正と負では逆転するわけです．このため正の数のときは左から0を入れるのですが，負の数のときはこれとは反対に1を左から入れます．

次にマシン語上ではどのように行っているか示します．例3，例4を御覧下さい．例3がＸ＝1．5のＩＮＴで，例4がＸ＝－1．5のＩＮＴです．どちらも単精度実数です．

例3　Ｘ＝1．5 ＩＮＴをとります．

X＝1.5	番地		切り捨て →		正規化 →	X＝1.0
$81	0074			$98		$81
11000000	0075			00000000		10000000
00000000	0076			00000000		00000000
00000000	0077			00000001		00000000
0×××××××	007C			0×××××××		0×××××××

※　×の所は何が入っていてもよい

例4　X＝－1．5のＩＮＴ

X＝－1．5

$81	0074
11000000	0075
00000000	0076
00000000	0077
1×××××××	007C

仮数部の
NEG
→

$81
01000000
00000000
00000000
1×××××××

切り捨て
→
上位桁から
1を入れる

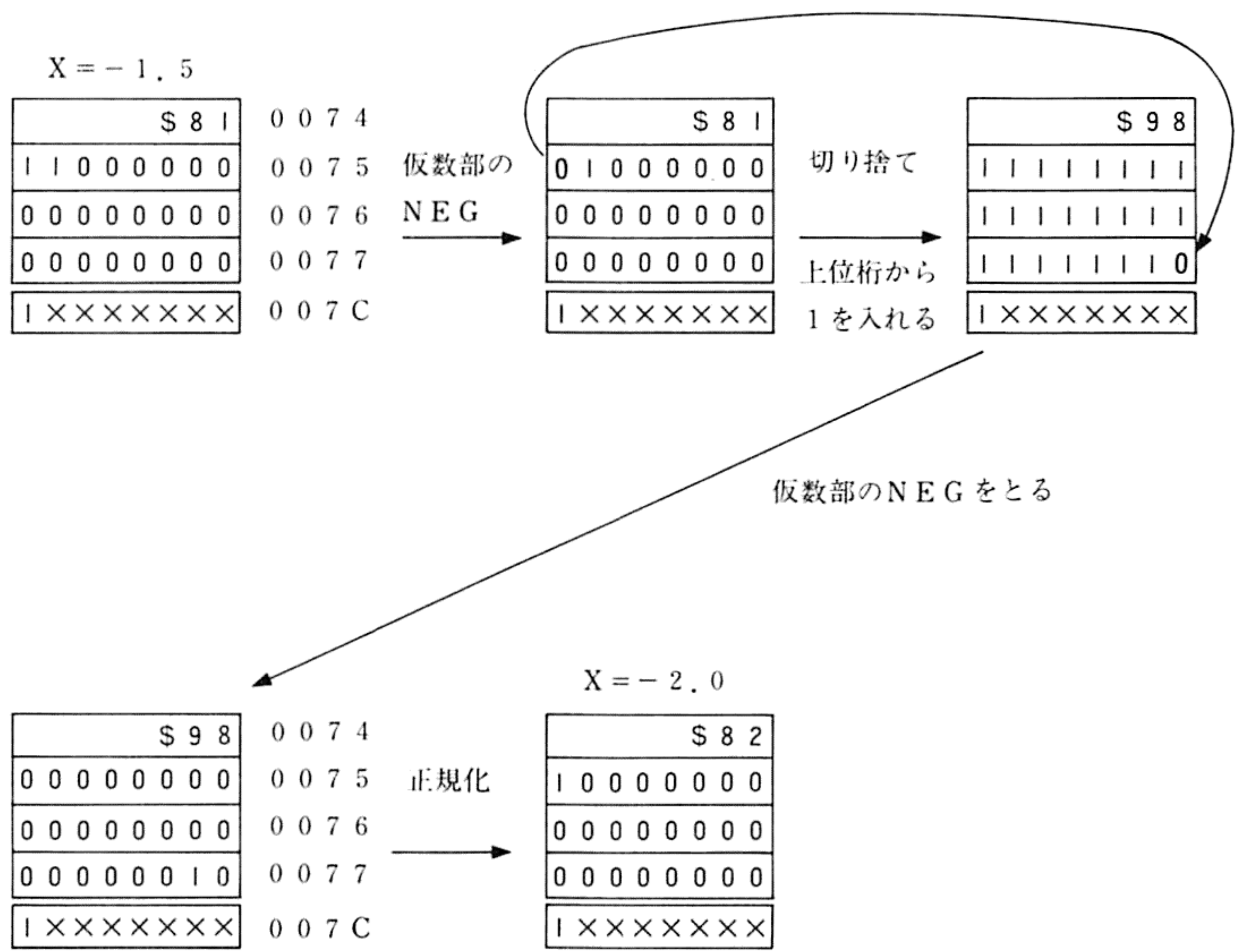
$98
11111111
11111111
11111110
1×××××××

仮数部のＮＥＧをとる

$98	0074
00000000	0075
00000000	0076
00000010	0077
1×××××××	007C

正規化
→

X＝－2．0

$82
10000000
00000000
00000000
1×××××××

※　×の所は何でもよい

5－4－9　ＦＩＸを求めるルーチン

アドレス	＄Ｂ４６１
機　能	引数の整数部分を求める．
レジスタ	Ａ，Ｂ，Ｘ，Ｕ
入力情報	ＦＡＣ１　＝引数
	（００１７）＝ＦＡＣ１の数値型を示す数値
復帰情報	ＦＡＣ１　＝引数の整数部分（数値型は引数のまま）
ＷＯＲＫ	（００７Ｃ）＝ＦＡＣ１の符号部
ＳＵＢ	＄ＢＣＢ３→数値型判断ルーチン　５－２－１参照
	＄Ｂ３Ａ０→数値の正負を判断する
	＄Ｂ４７７→ＩＮＴ　５－４－８参照
	＄ＢＤＣ９→符号反転ルーチン

解　説　ＦＩＸとＩＮＴとは次の関係にあります．

ＦＩＸ（Ｘ）＝ＳＧＮ（Ｘ）＊ＩＮＴ｛ＡＢＳ（Ｘ）｝

マシン語上では次のアルゴリズムを実行しています．

(1) 数値が整数型の場合はそのままリターンします．

(2) Ｘの符号によって以下のように場合分けします．

(i) 負のとき

符号を反転させ(3)へいきます．

(ii) 正のとき

そのまま(3)へいきます．

(3) ＸのＩＮＴをとります．

(4) (2)の(i)のときは符号を反転します．

※ (2)の(i)がＩＮＴと異なります．

５－４－１０　ＡＢＳを求めるルーチン

アドレス	＄Ｂ３Ｄ５
機　　能	引数の絶対値を求める．
レジスタ	Ａ，Ｂ
入力情報	ＦＡＣ１　　＝引数 （００１７）＝ＦＡＣ１の数値型を示す数値
復帰情報	ＦＡＣ１　　＝引数のＡＢＳ（数値型は入力時と同じ※）
ＷＯＲＫ	（００７Ｃ）
ＳＵＢ	＄Ｂ３Ｃ８→符号判定　５－２－２参照 ＄ＢＥ０３→負の整数を正の整数にする

解　　説　次のアルゴリズムを実行します（引数をＸとします）．

(1) Ｘが正のとき　そのままリターンします．

(2) Ｘが負のとき

(i) 実数型のとき

＄００７Ｃ番地（符号バイト）をクリアしてリターンします．

(ii) 整数型のとき

＄８０００（－３２７６８）以外のときはＮＥＧ（２の補数）をとってリターンします．

(iii) ＄８０００（－３２７６８）のときは絶対値が ３２７６８となって整数の範囲を越えてしまいます．ですからこの時に限り数値型を単精度実数の型にして３２７６８を表現します（この場合以外は※のようになります）．

５－４－１１ ＳＧＮを求めるルーチン

アドレス	＄Ｂ３ＢＥ
機 能	引数の符号を調べ正なら１，負なら－１，０なら０を与える．
レジスタ	Ａ，Ｂ
入力情報	ＦＡＣ１ ＝引数 （００１７）＝ＦＡＣ１の数値型を示す数値
復帰情報	ＦＡＣ１ ＝引数のＳＧＮ（整数型）
ＷＯＲＫ	ＦＡＣ１
ＳＵＢ	＄Ｂ３Ｃ８→符号判定 ５－２－２参照

解 説 ＄Ｂ３Ｃ８番地を呼び出すことによりＢレジスタに引数が負なら＄ＦＦ，正なら＄０１，０なら＄００が入ります．この値を整数としてＦＡＣ１に代入するのですが，整数は２バイトデータですからそのままでは代入できません．そこでＡレジスタを次のように換えてＤレジスタとして２バイトの整数を作り，これをＦＡＣ１に代入します．そのあと＄００１７に＄０２（整数型）を代入します．

ＦＡＣl	Ｂレジスタ	Ａレジスタ	Ｄレジスタ	l0進数
正	＄０１	＄００	＄０００１	１
負	＄ＦＦ	＄ＦＦ	＄ＦＦＦＦ	－１
０	＄００	＄００	＄００００	０

第6章 付　　録

6－1　ワークエリア一覧表

（０００００～０７７７）までのワークエリアの表です．００００～００ＦＦまでは上位２バイトを省略しました．使用法など詳しいことは本文を御参照下さい．なお，１つのエリアが複数の機能を持つ場合があるので注意して下さい．

アドレス	機　能　・　解　説
00 ～ 02	ＢＡＳＩＣ起動の際のＳＤの残骸
03 ～ 10	未使用
11 ～ 12	区切文字；カウンター；ＩＮＴの際の整数値保存用（11のみ）
13 ～ 14	編集中の行の長さ〔(13) 単独で「配列の次元」が入ることもある〕
15	倍精度フラグ　＄00　：単精度型 ＄00以外：倍精度型
16	ＤＩＭ文実行中・フラグ（変数サーチルーチンで使用）
17	ＦＡＣ１の数値の型
18	未使用
19	文字列領域不足フラグ（ガベージコレクション実行直後に立つ）
1A	添字評価禁止フラグ（変数サーチ）
1B	ＲＥＡＤ／$\overline{\text{INPUT}}$フラグ
1C	ＴＡＮ場合分けフラグ
1D	
1E ～ 1F	次のＳＡＣのポインタ
20	
21 ～ 22	現ＳＡＣのポインタ
23 ～ 24	Ｘレジスタの退避用（主にリターンアドレス）
25 ～ 26	〃
27 ～ 28	未使用
29 ～ 2F	ＦＡＣ３（仮数部）
30	未使用
31 ～ 32	ＳＰ（スタックポインタ）退避用など
33 ～ 34	テキストエリア先頭番地
35 ～ 36	単純変数 格納エリア先頭番地
37 ～ 3A	未使用
3B ～ 3C	配列変数 格納エリア先頭番地
3D ～ 3E	フリーエリア先頭番地
3F ～ 40	文字列スタック領域最終（上限）番地
41 ～ 42	文字列使用領域境界（ＳＳＰ：ストリングスタックポインタ）
43 ～ 44	文字列実効アドレス
45 ～ 46	文字列スタック領域先頭（下限）番地
47 ～ 48	実効行番号（現在実行中のプログラムテキスト行番号）
49 ～ 4A	ＣＯＮＴコマンド用再スタート行番号
4B ～ 4C	対サブルーチン情報受け渡し用レジスタ
4D ～ 4E	ＣＯＮＴコマンド用再スタート・テキスト番地

<table>
<tr><th>アドレス</th><th colspan="2">機　能　・　解　説</th></tr>
<tr><td>4F ～ 50</td><td colspan="2">実効・テキスト番地（現在実行中のテキスト番地，文を実行毎に更新される）</td></tr>
<tr><td>51 ～ 52</td><td colspan="2"></td></tr>
<tr><td>53 ～ 54</td><td colspan="2">DATA・RESTOREポインタ</td></tr>
<tr><td>55 ～ 56</td><td colspan="2">READ・INPUTコマンド読み込みポインタ</td></tr>
<tr><td>57</td><td colspan="2">変数名の長さ</td></tr>
<tr><td>58 ～ 59</td><td colspan="2">変数実効アドレス（変数本体の番地；変数サーチ上）；数値→文字列変換の文字数確認用</td></tr>
<tr><td>5A ～ 5B</td><td colspan="2">変数実効アドレス（変数本体の番地；代入文ルーチン上）</td></tr>
<tr><td>5C</td><td colspan="2">式の優先順位</td></tr>
<tr><td>5D</td><td colspan="2">FAC2の数値の型</td></tr>
<tr><td>5E</td><td colspan="2">比較演算子オーダ・レジスタ（b_0：>，b_1：=，b_2：<）</td></tr>
<tr><td>5F ～ 60</td><td colspan="2">逆向ブロック転送デスティネーション先頭（下限）</td></tr>
<tr><td>61 ～ 62</td><td colspan="2">逆向ブロック転送ソース先頭（下限）</td></tr>
<tr><td>5F ～ 62</td><td colspan="2">FAC1の単精度実数退避用</td></tr>
<tr><td>63</td><td colspan="2">文字列↔数値変換，小数点以下の数の保存</td></tr>
<tr><td>64</td><td colspan="2">文字列→数値変換の小数点フラグ；数値→文字変換のコンマカウンター</td></tr>
<tr><td>63 ～ 64</td><td colspan="2">逆向ブロック転送デスティネーション最終（上限）</td></tr>
<tr><td>63 ～ 66</td><td colspan="2">FAC1の単精度実数の退避用；VAL関数ルーチン内で（65）単独でも使用</td></tr>
<tr><td>67 ～ 68</td><td colspan="2"></td></tr>
<tr><td>69</td><td colspan="2">数値↔文字列変換の指数部操作用</td></tr>
<tr><td>69 ～ 6A</td><td colspan="2">逆向ブロック転送ソース最終（上限）</td></tr>
<tr><td>6B ～ 6C</td><td colspan="2">FAC1の単精度実数の退避用</td></tr>
<tr><td>6D ～ 6E</td><td colspan="2">FAC1の単精度実数の退避用；SAC番地の一時退避用（文字列加算で使用）</td></tr>
<tr><td>6F</td><td colspan="2">READ・キーボードINPUTフラグ</td></tr>
<tr><td>70 ～ 71</td><td colspan="2">READ・キーボード読み込みポインタ</td></tr>
<tr><td>72 ～ 73</td><td colspan="2">READ・キーボードによるINPUT時の読み込みポインタ退避用</td></tr>
<tr><td>74</td><td colspan="2">FAC1の指数部</td></tr>
<tr><td>75 ～ 7B</td><td colspan="2">FAC1の仮数部</td></tr>
<tr><td>7C</td><td colspan="2">FAC1の符号部</td></tr>
<tr><td>7D</td><td colspan="2">数値→文字列変換の符号フラグ</td></tr>
<tr><td>7E ～ 80</td><td colspan="2">一時的SAC</td></tr>
<tr><td>81</td><td colspan="2">切り捨て処理用レジスタ</td></tr>
<tr><td>82</td><td colspan="2">FAC2の指数部</td></tr>
<tr><td>83 ～ 89</td><td colspan="2">FAC2の仮数部</td></tr>
<tr><td>8A</td><td colspan="2">FAC2の符号部</td></tr>
<tr><td>8B</td><td>2項演算の演算フラグ</td><td rowspan="2">（8B：8C）として，文字列実効番地の一時退避用としても用いられる</td></tr>
<tr><td>8C</td><td>保護バイト</td></tr>
<tr><td>8D ～ 8E</td><td colspan="2">SDの位置；配列実効アドレス計算時の作業用アドレス</td></tr>
<tr><td>8F ～ 90</td><td colspan="2"></td></tr>
<tr><td>91 ～ 92</td><td colspan="2">FN関数の引数読み出し処理中のXレジスタ退避用</td></tr>
<tr><td>93</td><td colspan="2">WHILE～WENDカウンタ</td></tr>
<tr><td>94</td><td colspan="2">IF～ELSEカウンタ；INSTRカウンタ</td></tr>
</table>

アドレス	機　能　・　解　説
95	INSTRパラメータ（第1文字列の文字数）
96	INSTRパラメータ（第2文字列の文字数&カウンタ）
97 ～ 98	MID$文・第1引数の文字列変数実効アドレス
99	MID$文・文字数（第3引数）
9A	MID$文・文字位置（第2引数）
9B ～ 9C	FOR文用テキスト番地レジスタ
9D	AUTOフラグ
9E ～ 9F	AUTO開始行番号
A0 ～ A1	AUTO増分
A2 ～ A3	LIST・DELETE範囲先頭番地
A4 ～ A5	LIST・DELETE範囲最終番地
A6 ～ A7	ピリオド行番号
A8	FOR/$\overline{\text{NEXT}}$フラグ
A9	トレースモードフラグ（TRON/$\overline{\text{TROFF}}$）
AA ～ AB	FN関数内の読み込みポインタ（D9～DAと表裏ペアで使う）
AC	エラーコード
AD ～ AE	エラー発生行番号
AF ～ B0	スタックポインタ復帰用の値を保存するレジスタ
B1	エラーRESUMEフラグ
B2 ～ B3	RESUME再スタートアドレス
B4 ～ B5	ON　ERROR　GOTO行番号
B6 ～ B7	式の評価内でのテキスト番地退避用レジスタ
B8	PRINTUSING　小数点以下の文字数（小数点も含む）
B9	PRINTUSING　小数点以上の文字数
BA	PRINTUSING　FORMAT指定子
BB ～ BC	文字列アドレスの退避
BD ～ BE	PRINTUSING　処理レジスタ
BF	ファイル番号，同時にSFDを指すものとして使用される　｛$00→標準用 $11→システム用
C0	入力データ終了フラグ
C1	
C2	ファンクションコードPSET/PRESET/LINE文の文字コード
C3	WIDTH幅
C4	
C5 ～ C6	グラフィックX 座標
C7 ～ C8	グラフィックY 座標
C9 ～ CA	グラフィックX' 座標
CB ～ CC	グラフィックY' 座標
CD	フィールドの桁数
CE	フィールド改行桁数
CF	現在のポジション

アドレス	機　能　・　解　説
D0	改行桁数
D1	プロテクトフラグ
D2 ～ DD	汎用読み込みルーチン
D9 ～ DA	汎用読み込みポインタ
DE ～ E0	ＢＩＯＳジャンプルーチン
E1	乗算・除算の１バイト退避用
E2 ～ E3	配列の実効アドレス割り出し計算時の作業用レジスタ
E4 ～ E5	Ｄレジスター時退避用（テキスト作成ルーチン内）
E6	非実行文処理における，文字列定数内フラグ
E7 ～ E8	テキスト作成ルーチン内のＬＩＳＴアドレス
E9 ～ EA	ＶＡＬ関数実行中サイン――単独使用の（65）と関係あり
EB ～ EC	last line Ｘ座標
ED ～ EE	last line Ｙ座標
EF ～ F0	ＣＩＲＣＬＥ 中心のＸ座標
F1 ～ F2	ＣＩＲＣＬＥ 中心のＹ座標
F3 ～ F4	ＣＩＲＣＬＥ 半径
F5 ～ F6	ＣＩＲＣＬＥ 比率
F7 ～ F8	ＣＩＲＣＬＥ Ｘ座標の上限
F9 ～ FA	ＣＩＲＣＬＥ Ｙ座標の上限
FB	グラフィックカラーコード
FC	Line 文字コード
FD	ＣＩＲＣＬＥ 出力ポインタ
FE ～ FF	ＣＩＲＣＬＥ 終了位置
0100 ～ 017F	Display Sub System へのコマンド・パラメータをセットし復帰情報を得る領域
0180 ～ 01A1	TIME，DATE，CIRCLE，GET，PUT，RENUM，GCURSOL などのルーチンで用いられる汎用レジスタおよびHARDCのワークエリア（209バイト）
01A2	時刻設定のときの上位桁の保存用
01A2 ～ 01A3	インターバルのときの割り込み間隔
01A4	Line Character／Dot フラグ　キャラクタ…1　Dot…0
01A5	PSETフラグ：PSET…0　PRESET…1
01A6 ～ 01AB	割り込みの時刻・分・秒
01AC ～ 01AF	INTERVAL割り込みの20ｍｓ上での割り込み間隔
01B0	（HARDCで使用）
01B1 ～ 01B2	ＴＩＭＥ＄，ＤＡＴＥ＄の設定ルーチン内のＵレジスタの退避
01B3	シザリングを行うたびに＋１する
01B4	（HARDCで使用）

アドレス	機　能　・　解　説
01B5 〜 01BF	CHAINのオプション用レジスタ
01B5 〜 01C9	TIMERのREADで使用；SUBINデータ
01CA 〜 01D0	(HARDCで使用)
01D1 〜 01D3	SWI 3に対する割り込みベクトルの応答
01D4 〜 01D6	SWI 2に対する割り込みベクトルの応答
01D7 〜 01D9	SWIに対する割り込みベクトルの応答
01DA 〜 01DC	NMIに対する割り込みベクトルの応答
01DD 〜 01DF	IRQに対する割り込みベクトルの応答
01E0 〜 01E2	FIRQに対する割り込みベクトルに対する応答
01E3	プリンタ出力禁止コード
01E4	プリンタ出力禁止コード（LPT1：があるとする場合）
01E5	フォアグラウンドカラーコード
01E6	バックグラウンドカラーコード
01E7 〜 01E8	UNLIST行番号
01E9 〜 01EA	カセットギャップ出力定数：データの出力を行う前のモータの状態がオンなら$0A（10バイト），オフなら$FF（255バイト）だけギャップ（$FF）を出力
01EB	デフォルトデバイス番号　ROMモード…$02　DISKモード…$80
01EC 〜 01EE	“ON〜”の処理エントリ
01EF 〜 0216	2−3−4節の（4）の〔12〕の**表2・3・1**参照，予約語参照テーブル
0217 〜 0218	EXECの開始アドレス
0219 〜 021A	USR0の開始アドレス
021B 〜 022C	USR1〜USR9の開始アドレス（USR0と同様に2バイトずつに入る）
022D 〜 022F	ファイルディスクリプタのない“LOAD”のジャンプ命令
0230 〜 0232	“FILES”のジャンプ命令
0233 〜 0235	“GET”のジャンプ命令

アドレス	機能・解説
0236 ～ 0238	“PUT”のジャンプ命令（ディスク）
0239 ～ 023B	“KILL”のジャンプ命令（ディスク）
023C ～ 023E	“OPEN”のジャンプ命令（ディスク）
023F ～ 0241	“CLOSE”のジャンプ命令（ディスク）
0242 ～ 0244	1バイト出力のジャンプ命令（ディスク）
0245 ～ 0247	1バイト入力のジャンプ命令（ディスク）
0248 ～ 024A	ポジション読み出しのジャンプ命令（ディスク）
024B ～ 024D	“LOF／EOF”のジャンプ命令（ディスク）
024E ～ 0250	“INPUT”のジャンプ命令（ディスク）
0251 ～ 0253	“PEN”関係のジャンプ命令（リザーブ）
0254 ～ 02FF	“PEN”関係のジャンプ命令（リザーブ）が（0251～0253）のように3バイトずつ割り当てられる
0260 ～ 0262	ブロック入力ルーチン拡張用（＄DAA6より）
0263 ～ 0265	解読ルーチン拡張用（＄C63Dより）
0266 ～ 0268	解読ルーチン拡張用（＄C6D8より，BASICコマンドの拡張用）
0269 ～ 026B	MON処理系拡張用（＄ABF4より）
026C ～ 026E	CHAIN後処理サブルーチン拡張用（＄8432より）
026F ～ 0271	エラー処理ルーチン拡張用（＄8DD9より）；DISKモードで＄7E（JMP），＄7EB0と設定
0272 ～ 0274	単項式の評価ルーチン拡張用（＄91EDより）
0275 ～ 0277	1行翻訳ルーチン拡張用（＄C28E）
0278 ～ 027A	関数のサーチ，コールのルーチン機能拡張用（＄C622より）

アドレス	機能・解説
027B ～ 027D	1行逆翻訳ルーチン拡張用（＄C169より）
027E ～ 0280	関数のサーチ，コールのルーチン種類拡張用（＄C5E2より）
0281 ～ 0283	CHAIN後処理ルーチン拡張用（＄80F6より）
0284 ～ 0286	TERMルーチン拡張用（＄D52Dより）
0287 ～ 0289	文字列代入ルーチン拡張用（＄91B7より）；DISKモードで＄7E（JMP）＄7FC4と設定
028A ～ 028C	文字列がバッファ領域のものかどうか判別するルーチン拡張用（＄C74Fより）
028D ～ 028F	Abortルーチン拡張用（＄D678より）
0290 ～ 0292	Breakチェックルーチン拡張用（＄DB59より）
0293 ～ 0295	変数名ロードルーチン拡張用（＄9519より）
0296 ～ 0298	文字領域確保ルーチン拡張用（＄97FDより）：領域が不足したときの処理拡張用
0299 ～ 029B	ファイル名設定・デバイス選択ルーチン拡張用（＄CC3Fより）
029C ～ 029E	IRQの割り込み（外部からのもの）ルーチン拡張用（＄D319より）
029F ～ 02A1	1行翻訳ルーチン拡張用（＄C502より）：コマンド後処理の拡張用
02A2 ～ 02AB	TAB設定テーブル
02AC ～ 02AD	Input バッファポインタ（未使用）
02AE ～ 02AF	未使用
02B0 ～ 02B1	カセット用バッファポインタ　カセット及びPLAY文で用いられるバッファの先頭アドレス
02B2 ～ 02B3	デバイス分類テーブルの先頭アドレス
02B4 ～ 02B5	システムファイルディスクリプタの先頭アドレス
02B6 ～ 02B7	PEN割り込みテーブルの先頭アドレス（未使用）

アドレス	機　能・解　説
02B8 〜 02B9	COMn割り込みのテーブル先頭アドレス
02BA 〜 02BB	KEY割り込みのテーブル先頭アドレス
02BC 〜 02BD	TIME割り込みのテーブル先頭アドレス
02BE 〜 02BF	INTERVAL割り込みのテーブルの先頭アドレス
02C0 〜 02C9	COMnポートポインタの登録テーブル
02CA 〜 02CB	未使用
02CC 〜 02CD	カセット入出力先頭アドレス
02CE	カセット・ブロックのタイプ
02CF	カセット・ブロックの長さ
02D0	カセット使用フラグ
02D1 〜 02D2	カセット・バッファ中のポインタ
02D4	カセット・テープファイルモード
02D5	カセット・エラーフラグ
02D6	カセット・モータスイッチフラグ
02D7	カセット・モータ状態フラグ
02D8	カセット・ファイル名の画面出力フラグ
02D9	LF出力禁止フラグ
02DA	ファイルモード　$10…インプット　$20…アウトプット　$40…ランダム
02DB	ファイルタイプ　$00…BASICソース　$01…BASICデータ　$02…マシン語ファイル
02DC	アスキーフラグ　$00…バイナリ　$FF…アスキー
02DD	ファイル名の長さ
02DE 〜 02E5	ファイル名
02E6	デバイス番号
02E7 〜 02ED	ファイルオプション
02EE	LOAD後即実行フラグ
02EF	MERGE指定フラグ
02F0	LOADMフラグ
02F1 〜 02F2	LOADMオフセット値
02F3	LOAD（バイナリ）実行中フラグ

アドレス	機　　能　・　解　　説
02F4 〜 02FF	時刻、分、秒、年、月、日　0〜9までの10進数が2桁ごとに入る
0300 〜 030A	未使用
030B	カーソルのX座標（キャラクタ）
030C	カーソルのY座標（キャラクタ）
030D	画面行数
030E	スクロール開始行
030F	スクロール終了行
0310	PF・キー表示行数
0311	フォーマット改行桁数（スクリーン）
0312 〜 0313	ブレーク・キー押下フラグ
0314 〜 0315	MONのDおよびMコマンド・リード／ライト・アドレス
0316 〜 0317	MON処理におけるSP退避用
0318 〜 031B	RNDの演算用レジスタ
031C 〜 031E	RNDの処理用レジスタ
031F 〜 0338	変数名のDEF型指定用テーブル
0339	リンクポインタ領域確保用のダミーコード
033A 〜 033B	テキストバッファで編集中の行番号
033C 〜 043B	テキストバッファ
043D 〜 053C	システム入(出)力用バッファ
053F 〜 0567	数値→文字列変換における文字列バッファ
0568 〜 0577	変数名参照用テーブル
0578 〜 0595	文字列作業領域（SAC1〜SAC10）
0596 〜 0597	READ／INPUT処理系におけるポインタ（D9：DA）の一時退避用
0598	エンディングフラグ
0599	INPUT／LINEINPUTフラグ
059A	待機中のBASIC割り込みの個数（セマフォ：Semaphore）

アドレス	機　　能　・　解　　説
059B	TERM識別フラグ
059C	ターミナルモードでの通信モード（全二重／半二重）
059D ∫ 059E	解放RAM領域の終わり
059F	ROMモード／DISKモード識別フラグ ROM…$00　DISK…$FF
05A0 ∫ 05A7	RCB　BIOSへのRCB
05A8	コンソールコントロールフラグ
05A9	インプット継続フラグ
05AA ∫ 05AB	LPT0：プリンタのジャンプテーブル先頭アドレス（桁数などの読み込み用）
05AC	プリンタONフラグ
05AD	単色表示　0…カラー　$FF…単色
05AE	PF・キー割り込みのPF・キー番号
05AF ∫ 05B0	PF・キー割り込み制御フラグ
05B1 ∫ 05B2	TERMモードでのPF・キー割り込みの実行ルーチンのアドレス （TERMモードでないときは$0000が入る）
05B3 ∫ 05B4	漢字のX座標
05B5 ∫ 05B6	漢字のY座標
05B7	1行入力にフィールド設定をするときにイレーズするかどうか決定する
05B8 ∫ 05BB	未使用
05BC	CHAIN実行中フラグ
05BD	TERMモードでPF10の処理のときのカウンタの初期値
05BE	プリンタ継続フラグ（改行しないために使われる）
05BF	プリンタ自動改行フラグ
05C0	PLAY文実行中フラグ
05C1	オシレータ1のテンポ　（PSG）
05C2	オシレータ1のオクターブ　（〃）
05C3	オシレータ1の指定していない音符の基本長　（〃）
05C4	オシレータ1の休符基本長　（〃）
05C5	オシレータ1のボイスまたはエンベロープパターン　（〃）
05C6 ∫ 05CA	オシレータ2のディスクリプタ　オシレータ1（05C1～05C5）のように使用
05CB ∫ 05CF	オシレータ3のディスクリプタ　オシレータ1（05C1～05C5）のように使用
05D0 ∫ 05D1	PLAY文中でのエンベロープ周期
05D2	割り込み（IRQ）のマスクレジスタ

アドレス	機　能・解　説
05D3 〜 05D5	"BUBINI" のジャンプ命令
05D6 〜 05D8	"BUBW" のジャンプ命令
05D9 〜 05DB	"BUBR" のジャンプ命令
05DC	COMn識別フラグ
05DD	アクティブVRAMコード
05DE	ディスプレイVRAMコード
05DF 〜 05E1	タイマの割り込みのジャンプ命令
05E2 〜 05E4	KEY・INの割り込みのジャンプ命令
05E5	FATへの記入レベル
05E6 〜 05EC	未使用
05ED 〜 06EB	カセット用バッファ（PLAY文でも使われる）
06EC 〜 06ED	KYBDジャンプテーブルポインタ
06EE 〜 06EF	SCRNジャンプテーブルポインタ
06F0 〜 06F1	CAS0ジャンプテーブルポインタ
06F2 〜 06F3	プリンタジャンプテーブルポインタ
06F4 〜 06FD	COMnのジャンプテーブルポインタを2バイトずつ格納
06FE 〜 070B	将来入出力のデバイスを拡張したときのジャンプテーブルポインタが入る
070C 〜 071D	SFD（システムファイルディスクリプタ）テーブル
071E 〜 0722	PEN割り込みテーブル
0723 〜 073B	COMn割り込みテーブル×5（5バイトずつ）
073C 〜 076D	KEY割り込みテーブル×10（5バイトずつ）
076E 〜 0772	TIME割り込みテーブル
0773 〜 0777	INTERVAL割り込みテーブル

6－2　メモリマップ

この表には，データブロックを除いて，主にエントリ番地のみを掲載しました．1つの処理系に対するエントリポイントがいくつも続くような場合には，適宜段付けを行っています．

アドレス	機能・解説
8000	初期設定ルーチン
8013〜802F	転送用データ
8030〜804D	予約語リスト
804F〜8059	ジャンプテーブル
805A	拡張コマンド用・解読実行ルーチン
8077	関数予約語サーチ・コール・ルーチン
808E	CHAIN前処理メイン
80EB	CHAIN後処理メイン
8131	CHAINオプション読み込みルーチン
81A7	CHAINサブルーチン群スタート番地
83BF	CHAIN後処理サブルーチンならびにエラー処理ルーチン
8435	CHAIN後処理系サブルーチン
8448	ERASE
848B	コールドスタート
73	COMn設定
8597	ブロック転送ルーチン（Bレジスタ分）
859F	A・Xレジスタ転送ルーチン
8656	コンソール初期化ルーチン
8678〜867A	先行入力禁止データ
867B	BIOSイニシャライズルーチン
8684	ホットスタート
92	割り込みベクトル設定ルーチン
A6	TAB設定ルーチン
CC	PF・キーイニシャライズルーチン
8703〜8757	電源投入時のメッセージ
8758	NULL（$00）
8759〜876A	COMnのジャンプテーブルのデータ
876B〜87B4	PF・キーのデータ

アドレス	機能・解説
87B5〜87C0	$D2ルーチンの内容
87C1〜87D2	割り込みベクトルの応答(01D1〜01E2)転送用の内容
87D3〜87E8	ワークエリア（01E3〜01F8）転送用の内容
87E9〜87FD	プリンタジャンプテーブル転送用の内容
88FE〜8803	デバイス分類テーブルの（06EC〜06F1）転送用の内容
8804〜880F	割り込みベクトル転送用の内容
8810〜8815	年・月・日（82年12月1日）（02FA〜02FF）転送用の内容
8816〜886B	標準関数のジャンプテーブル（SGN INT………DATE）
886C〜888F	2項演算子の優先順位テーブル及びジャンプテーブル（ベキ乗のみ注意）
8890〜8A2B	基本コマンド及び2項演算子の予約語綴りテーブル
8A2C〜8AD8	基本関数の予約語綴りテーブル
8ADA〜8B71	基本コマンドジャンプテーブル
8B72〜8D22	基本予約語索引テーブル
8D23〜8D56	基本予約語索引用のアルファベット順見出しテーブル
8D57〜8D60	「Ready」メッセージ文字列
8D61〜8D68	「Break」メッセージ文字列
8D69	スタック上のネスティング検索①
(8D6B)	同　上②
8D93	逆向ブロック転送（メモリフルテスト付）
8D95	逆向ブロック転送（メモリフルテストなし）

アドレス	機　能　・　解　説	アドレス	機　能　・　解　説
8DAA	メモリフルテスト1	9162	行番号読み込みルーチン（文字列→行番号）
8DAE	メモリフルテスト2	9195	LET──代入文処理──（メイン）
8DBC	X+B→X	B3	LET（文字列代入）
BD	X+D→X	91E8	Missing Operand エラーエントリ
BF	（23：24）+D→X	91ED	単項式評価ルーチン
8DC6	Out of Memory エラー	9288	▼（▼＋式の評価＋▼）▼
8DCF	Direct Statement In Fileエラー	928C	右カッコ▼）▼チェック
	エントリ	928F	左カッコ▼（▼チェック
8DD1	エラー処理ルーチン・エントリ	9292	コンマ▼，▼チェック
8E63	メイン・プロンプト表示部	9294	任意の文字コードのチェック(B-reg)
	(Break・Error・Abort 用)	92A0	Syntax Error エントリ
72	(「Ready」出力)	92BE	式の評価＋文字型エラーの型判断
8E88	テキスト作成ルーチンエントリ	92C3	式の評価
8F1C	行番号サーチルーチン	9364	FAC1⇔FAC2 { エントリ 9364一倍精度用, 9374一単精度用, 9384一整 数 用 }
8F32	NEW (1)〔NEWコマンド・エントリ〕	93CD	除算　/
39	NEW (2)	93E8	PSHS FAC1
4B	NEW (3)	F4	PSHS FAC1（FN関数用）
51	NEW (4)〔CLEAR文相当〕	9411	PULS FAC2　{ OR──$94AD
70	NEW (5)	9433	比較演算メイン　XOR──94B2
82	NEW (6)	6B	文字列の比較演算　EQV──94B7
91	NEW (7)	94A8	AND, OR, XOR, EQV, IMP……　IMP──94BC }
8FBC	RESTORE	94C5	DIM（配列変数の登録）
C9	RESTORE 初期化	94CE	変数名→変数名参照用テーブルにロード
8FD0	ENDルーチン先頭	94FD	アルファベットチェックルーチン
D7	Break エントリ	9506	数字チェックルーチン
D9	STOP	950F	変数サーチ（変数検索，読み出し，登録）
E4	暗黙END用エントリ	(9570)	新変数登録ブロック・エントリ
9002	CONT	95AF	変数検索ルーチン
9012	RUN	C7	変数識別子作成ルーチン
902D	GO	95D9	正整数データ評価ルーチン（式の評価付）
39	GOSUB	DE	〃　（〃　無）
55	GOTO	95E6	配列変数サーチ（検索，登録など）
9071	RETURN	(966E)	新・配列変数登録ブロック・エントリ
9081	Return Without Gosub エラー	(96BC)	配列要素の実務アドレス割り出しブロック
9084	Undefined Line Number エラー	9663	Illegal Function Call エラー
90A0	DATA	972C	FRE関数
90A3	REM及びELSE	974C	1バイト整数・最終評価&FAC1に格納
90A8	非実行文の処理1	4D	整数型データ・　〃
90AB	非実行文の処理2		※上の2つのルーチンでは，Dレジスタ
90F1	IF～THEN～ELSE		⇒FAC1 と整数型宣言が行われる
9134	ON（式）およびON ERROR	9752	STR$関数
913A	ON（式）GO	9760	文字列データ長の読み出しルーチン

アドレス	機能・解説	アドレス	機能・解説
9790	文字列格納領域確保，一時的SAC格納	9A02	式の評価＋整数化→Xレジスタ
94	一時的にSAC（7E〜80）にSDを格納	9A05	FAC1→整数化→Xレジスタ
		9A22〜9A25	単精度型定数「65536」
9799	文字列定数などの読出し評価ルーチン1		
9B	〃 2	9A26	PEEK関数
9D	〃 3	9A30	POKE文
9F	〃 4	9A3A	行番号定数の評価ルーチン
A6	文字列格納領域確保&最終評価ルーチン	9A50	TAB（ ）関数の処理
97AD	文字列式の最終評価ルーチン（SACに結果を格納；文字列型関数の出口ルーチンに当たる；SACに番地→FAC1）	9A7F	LPRINT
		9AB3	PRINT
		9AC0	PRINT実行ルーチン
97D7	文字列格納用の領域を確保するルーチン	9B47	X座標が0でなければ改行する
980D	ガベージコレクション・エントリ	9B50	改行ルーチン
9850	文字列ゴミ化ルーチン	9B68	PRINT文中の「，」の処理
9865	文字列リンクポインタの修正ルーチン	9B7E	PRINT文中の「；」の処理
987C	文字列型配列変数・転送後の文字列リンクポインタ修正ルーチン	9B9F	出力がプリンタかどうかチェック
		9BB5	SPC（ ）関数の処理
98AE	文字列加算（＋）	BE	空白出力処理ルーチン
98E4	文字列転送・代入1（SDにより）	9BD4	空白出力実行ルーチン
E6	〃 2（Xにより）	9BDB	1行出力ルーチン
98F1	文字列実効番地読み出し&SACつぶし（式の評価から）	F3	同上（PRINT文などから）
		F6	同上（SACを考慮しないもの）
F4	同上（型判断から）	9C22	空白を1つ出力するルーチン
F7	同上（FAC1読み出しから）	9C25	「？」を出力するルーチン
F9	同上（文字数読み出しから）	9C2B	PRINT@（漢字）実行ルーチン
991B	SACつぶし1	9CD4	INSTR関数
1F	〃 2	9D59	STRING$関数
992A	LEN関数	9D7B	SPACE$関数
2F	文字列実効番地読み出し&文字数テスト	(9D91)	MID$文用サブルーチン
9933	CHR$関数	9DC1	MID$文
9947	ASC関数	9E2C	HEX$関数
9952	LEFT$関数	2D	OCT$関数
996F	RIGHT$関数	(9E76)	HEX$，OCT$用サブルーチン
9979	MID$関数	9E7E	&定数 } 9EBA〜9ECB 右の定数の処理のためのサブルーチン
9C	MID$関数用サブルーチン	94	&O定数
99BA	式→1バイト整数データ→Breg ①	A5	&H定数
BC	〃 ②	9ECC	（4B：4C）→行番号として出力
BF	〃 ③	CE	Dレジスタ →行番号として出力
99CA	VAL関数	9ED1	LIST出力ルーチン・エントリ
99F9	コンマで区切られた引数の評価〔整数化→（4B：4C）；Breg〕	9F0B	LIST・DELETE範囲の検索
		9F5C	文の終結チェック（構文チェック）
FD	コンマ以降の式の評価→Breg	9F62	LISTなどの行番号評価ルーチン

アドレス	機　能　・　解　説	アドレス	機　能　・　解　説
9F73	DEFSTR	A9CA	RENUMメインルーチン（〜AA2D）
76	DEFINT	AA2E	RENUM—パス1
79	DEFSNG	2F	RENUM—パス3
7C	DEFDBL	AAA8	RENUM—パス2（行番号スタック）} ユーザは，このルーチンを独立にコールすることができる
9FBC	DELETE		
D3	テキスト一部消去（DELETE用）		
9FE7	AUTOコマンド処理系	AB73	パス—3用サブ（行番号付け直し）
A01A	TRON	AB9A	RENUM—パス4（エラー検出）
1B	TROFF	ABF4	MONコマンド処理
A07F	PRINT USING（〜A202）		メインルーチン（〜AC2E）
A203	FOR エントリ } 〜A36C	AC2F	Gコマンド
A2C7	NEXTエントリ } 〜A36C	AC37	16進4桁（2バイト）出力 by Dreg
A36D	WHILE・WEND用	3D	16進2桁（1バイト）出力 by Areg
	ネスティング・対応チェックルーチン	AC54	16進2桁入力→B-reg（1バイト）
	（WHILE・WENDの数を	5F	MON用1行入力ルーチン
	カウントする；〜A43E）	6C	16進4桁入力→D-reg（2バイト）
A43F〜A445	PRINT USING処理系の飛び地	AC95	Mコマンド
		ACC4	Dコマンド
A446〜A70E	エラーメッセージ群（41種類）	ACFB	Rコマンド
	〔ROMモードのもののみ〕	AD36〜AD46	レジスタ名を表す文字列群
A70F	ON ERROR GO エントリ		「CC、A␣、B␣、……、PC」
A739	ERROR文エントリ	47	レジスタ名出力ルーチン
A744	RESUME	AD54	WHILE
A784	SWAP	AD81	WEND
A7A9	Xレジスタの指す変数⇔Uレジスタの指す変数〔変数交換ルーチン〕	B1	スタック上のネスティング情報の検索（WHILE用）
C4	文字列変数のリンクポインタ交換	ADD7	Wend Without While エラー
A7D9	DEF	ADDC	HARDC
DF	DEF FN文処理ルーチン	ADF0	HARDC0実行ルーチン
FA	FN関数名登録ブロック	AE69	HARDC1実行ルーチン
A80E	FN関数名サーチルーチン	AE8C	HARDC2実行ルーチン
A81F	▼Illegal Direct▼のチェック	AEAF	HARDC用アクティブVRAM変換ルーチン
A82A	FN関数処理系（〜A951）		
(A892)	(AA:AB)⇔(D9:DA)	AEB7	HARDC用アクティブVRAM逆変換ルーチン
(A925)	文字列型引数の評価ルーチン		
A952	VARPTR関数	AEBC〜AEC9	HARDC1のときディスプレイVRAMコードによって設定される黒印字カラー指定バイトと灰印字カラー指定バイトのデータ（2バイトずつ）
A969	DEF USR文処理ルーチン		
80	USRジャンプテーブル番地の読み込み		
A9A0	USR関数		
A9CA	RENUMコマンド・エントリ	AECA〜AED1	HARDC2のとき同様に設定される黒印字カラー指定バイトのデータ
	RENUM処理系サブルーチン群は$A9BD〜$ABF3の範囲にある	AED2	ANPORT

アドレス	機能・解説	アドレス	機能・解説
AF0C	FAC1+0.5	B615	Dreg 内の整数出力〔行番号出力用〕
AF11	実数減算ルーチン	B618	符号無し整数の出力〔行番号出力用〕
AF1A	実数加算ルーチン	B61B	実数の出力（符号なし）
AF97	正規化ルーチン	B620	数値→文字列変換ルーチン
B00A	丸めルーチン	B622	数値→文字列変換ルーチン(フォーマット付)
B044	Overflowエラーエントリ	BA62~ED	：数値→文字列変換用データ
B068	切り捨てルーチン	BAEE	SQR（平方根）
B089 ~ B08C	データ 1.0000	BAF7	ベキ乗ルーチン
B08E ~ B0A9	LOG演算データ	BB36 ~ BB5A	EXP演算データ
		BB5B	EXP
B0AA	LOG（自然対数）	BB94	多項式1
EE	実数乗算ルーチン	BBA3	多項式2
B1BA	メモリ上からFAC 2に数値格納	BBE5	RANDOMIZE
B1DF	汎用指数部演算ルーチン	BBFE	RND（乱数）
B203	Overflow エラーエントリ	BC4E ~ BC73	「Random Number Seed」データ
B206	FAC1*10		
B219 ~ B21F	データ10.0	BC74	CINT（整数化ルーチン）
		BCA5	FAC1•型判断（文字列型以外エラー）
B221	Division By Zero エラーエントリ	BCA9	Type Mismatch エラーエントリ
B226	FAC1/10	BCAE	FAC1•型判断（文字列型エラー）
B234	実数除算ルーチン	BCB3	FAC1•型判断（エラーなし）
B2EC	FAC3→FAC1（仮数部のみ）	BCC1	数値型の強制変換（Aregにオーダ）
B301	メモリからFAC1へ数値転送（Xreg）	BCCD	CDBL（倍精度化ルーチン）
B32B	FAC1のワークエリア退避（B330も）	BCD5	単精度型→倍精度型変換ルーチン
B335	FAC1→（5A：5B)が指すメモリ領域	BCE0	CSNG（単精度化ルーチン）
B337	FAC1をメモリに数値転送（Xreg）	BCEE	倍精度型→単精度型変換ルーチン
B35F	FAC2→FAC1数値転送	BCFF	整数型→単精度型変換ルーチン
B380	FAC1→FAC2数値転送	BD19	(76：77）符号無整数を単精度化 行番号定数用
B3A0	FAC1（実数型）の符号判定	BD29	整数•実数判断ルーチン（演算用）
B3BE	SGN（符号）	BD3F	減算 −
C0	Bレジスタを整数化	BD44	整数同士の減算
C8	符号判定ルーチン	BD4D	加算 +
B3D5	ABS（絶対値）	BD52	整数同士の加算
B3E1	FAC1, FAC2比較ルーチン（実数）	BD73	乗算 *
B3EC	FAC1, メモリ比較ルーチン（実数）	BD78	整数同士の乗算
B435	切り捨てルーチン2	BDC9	FAC1の符号反転
B461	FIX	BE0C	¥（整数除算）；BE33 MOD
B472	INT	BE44	数値比較演算（<, =, >）
B506	文字列→数値変換（倍精度基本）	BE60	COS（余弦関数）
B50B	文字列→数値変換（整数基本）	BE66	SIN（正弦関数）
B60E	「In␣(行番号)」の出力	BEB1	TAN（正接関数）

アドレス	機　能　・　解　説
BED4〜BEF4	SIN, COS, TAN演算データ
BEF5	ATN（逆正接関数）
BF23〜BF47	ATN演算データ
BF48	LINEINPUTエントリ
BF71	INPUTオーダ読み込みルーチンエントリ
72	LINEINPUT ルーチン エントリ
BFB0	INPUT/LINEINPUT用 1行入力ルーチン・エントリ
BFC0	INPUT入力ミスの処理ルーチン
BFDD	INPUTコマンド・エントリ
F3	READエントリ
F7	READ/INPUT共用 メインルーチンスタート（〜C155）
(C0F4)	定数スキップルーチン（READ用）
(C10E)	DATA文サーチ（READ文用）
(C132)	1パス/2パス制御ブロック
(C13A)	2パスの場合のパス1エンディング
(C14D)	READ/INPUTエンディングブロック
C156〜C168	「?Redo From Start」メッセージ
C169	1行逆翻訳ルーチン（〜C287）
C288	1行翻訳ルーチン（〜C5E1）
C490	アルファベットテストルーチン （小文字⇒大文字変換付き）
C495	英小文字⇒大文字変換ルーチン
(C528)	1行翻訳・行番号定数処理ルーチン
C5E2	標準関数コールルーチン（〜C63C）
C63D	解読実行ルーチン（メイン）（〜C6EC）
5D	エラー処理ルーチンからのエントリ
7A	ダイレクト実行用エントリ
(C680)	解読実行ルーチン（サブ） ※各コマンド処理系へ分岐する場所
(C6DB)	暗黙END処理ブロック
C6ED〜C72F	省略形キーワード・綴りテーブル
C730	リンクポインタ修正ルーチン
C750	バッファ領域の文字列かどうかを判断
C760	数字チェック＆空白スキップ （$D2, D8ルーチンの延長）

アドレス	機　能　・　解　説
C76F	EXEC文エントリ
C77B	CLEARコマンドエントリ
BB	Out Of Memoryエラーエントリ
C7BE	式の評価→整数→Breg（上限値付）
C7CA	COLOR
C7F1	COLOR=（パレット, カラー）
C814	CONSOLE
C85F	式の評価→整数スタック（"，"のチェックも行う）
C87B	WIDTH
C8BC	画面設定ルーチン
C8EA〜C8EF	画面設定ルーチン用のBIOSへの RCBデータ
C8F0	LOCATE
C912	UNLIST
C923	CSRLIN
C92A	POS関数
C947	LPOS関数
C953	FIRQ
C9D8	行番号定数評価→サーチ→番地格納 （「ON〜割り込み」用の行番号サーチ）
C9E8〜C9F9	CAS0用のジャンプテーブル
C9FA	カセットEOF
CA02	カセットLOF
CA06	カセット　オープンルーチン
18	モード"O"処理
70	カセット用ポインタの初期化ルーチン
73	モード"I"処理
78	ファイルの属性の設定
CA8D	カセット1ブロック入力（デバイス→カセット用バッファ）ルーチン
A4	カセット用ポインタの設定
CAAE	Device In Use エラー
CAB1	Device I/O Error エントリ
CAB6	カセットクローズ　ルーチン
BA	アウトプットモード処理
DD	カセット終了処理
E0	カセット使用フラグ解除
CAE4	LOAD?
E6	SKIPF
F7	エンドブロックまでの読み飛ばし

アドレス	機　能　・　解　説	アドレス	機　能　・　解　説
CB02	カセット1バイト入力（カセット用バッファより）ルーチン	CDA6	プロテクトセーブ実行，及びプロテクト逆変換ルーチン
CB1E	カセット1バイト出力（カセット用バッファへ）ルーチン	AA	プロテクト逆変換ルーチン
CB36	カセット1ブロック出力（データブロック；カセット用バッファから）ルーチン	CDC0	PEEK，POKE用のプロテクトチェックルーチン
38	同上（任意のタイプのブロックとして）	CDC6	プロテクトチェックルーチン
CB4D	カセット　ポジションチェック	C8	RENUMからのUNLISTエラー
CB57	プログラムまたは，データファイルのヘッダ読み込みルーチン	CA	Protected Programエラー
59	同上（画面表示を行わせない場合）	CDCF	バイナリセーブルーチン
95	メッセージ画面出力ルーチン	D6	バイナリセーブ（プロテクト用）
A6	メッセージ及びカセット上のファイル名画面出力ルーチン	CDF0	式の評価→整数→スタック（2バイト）
AD	カセットのファイル名画面出力ルーチン	CDFC	2バイト出力ルーチン（SAVEM用）
CBB5	FILES	FE	1バイト出力ルーチン（$CDFC用）
CC14～CC24	「Searching」のデータ	CE04	SAVEM
CC25～CC2C	「Found：␣」のデータ	CE4B	クローズ実行ルーチン
CC2D～CC33	「Skip：␣」のデータ	4D	同上（ファイル番号を指定）
CC34	CC37の「FILES」用エントリ	CE73	CLOSE
CC37	ファイル名・デバイス番号設定ルーチン	CE81	すべてのファイルをクローズする
3C	同　上 「LOAD？」「SKIPF」用のエントリ	CE8D	CE90に同じ（▼，▼のチェック付）
AC	ファイル名の転送処理	CE90	ファイル番号設定ルーチン
C5	デバイス番号設定処理	CEA3	Bad File Number エラー
CCFC	Bad File Descriptor エラー	CEA8	OPEN
CD01	LISTエントリ	CEDC	ファイル　オープンルーチン（インプットモード）
CD0E	プログラムファイル用のファイル名・デバイス番号設定ルーチン	DF	同上（アウトプットモード）
CD20	アスキーセーブ前処理ルーチン	E1	同上（任意のモード）
24	アスキーセーブ／LIST共通前処理ルーチン	CEF4	Bad File Mode エラー
CD3E	SAVEエントリ	CEF7	File Already Open エラー
5B	プロテクト変換処理	CEFC	ディスク以外のデバイスのオープン処理
CD71	プロテクト変換用初期設定	CF0A	Device Unavailable エラー
CD79	プロテクト変換ルーチン1	CF22	ロード（「RUN"FD"」エントリ）
80	プロテクト変換ルーチン2	29	ロード（「MERGE」エントリ）
CD88	LLIST	31	ロード（「LOAD」エントリ）
CD96	プロテクト変換ルーチン3	77	ロード（「CHAIN」エントリ）
		A3	ロード前処理ルーチン
		C8	ロード（バイナリ）前処理ルーチン
		D3	バイナリロード実行ルーチン
		D011	ロード後処理ルーチン
		D027	1バイト入力（エラー付）ルーチン
		D030	LOADM
		D06B	2バイト入力ルーチン
		6D	1バイト入力（$D06B用）ルーチン

アドレス	機能・解説	アドレス	機能・解説
D072	1バイト入力ルーチン	D475	ON処理
D08E	1バイト出力ルーチン	84	STOP処理
D0CA	ポジション・デバイスチェックルーチン	D48A	BASIC割り込みが発生した割り込み
D0ED	ファイルチェックルーチン		処理ルーチンを待機させる
D0FA	ファイルモードチェック(アウトプット)	D498	BASIC割り込みの初期化ルーチン
FD	ファイルモードチェック(インプット)	D4B0	BASIC割り込みの初期化ルーチンの
D11A	EOF	～D4B5	BIOSへのRCBデータ
1D	LOF	D4B6	サブシステムのSET TIMER用の
D150	ファイル番号(1～16)チェック	～D4BA	コマンド及びパラメータ
53	同上(式の評価なし)	D4BB	TERMモードでのPF・キー
D15C	INPUT$関数		割り込み処理ルーチン
D198	COMn クローズルーチン	D4BF	TERMモードでの
D1AD	COMn オープンルーチン		PF・キーの処理ルーチン
BA	オプション処理	D4CF	PF6 の処理
D20C	オープン実行(D1AD中)	D3	PF7 の処理
38	8251A(USART)設定ルーチン	D7	PF8 の処理
D250～D25B	オプションのデータ値	DF	PF10の処理
		D4F7	111 ms 時間待ちルーチン
D25C	COMn 1バイト出力ルーチン	D50E	TERMモードでのPF・キー処理の
9F	1バイト送信ルーチン	～D517	ジャンプテーブル
D2AA	送信イネーブル待ちルーチン	D518	TERMのオプション転送ルーチン
D2B5	COMn 1バイト入力ルーチン	D523	プットウエイト禁止ルーチン
D2CD	Buffer Overflow エラー	D528	プットウエイト選択ルーチン
D2FC	IRQ	D52D	TERM
D39C	LPT0 及びCOMnのポジションチェ	D5A2	TERMメインルーチン
	ック	D5D6	TERM処理0
D3A5	COMn EOF	D5E3	TERM LOFチェック
D3B4	COMn LOF	D5EC	TERM処理1
D3C2	BASIC割り込みの制御ルーチン	D5F0	TERM処理2
D407	ON COMへのエントリ	D5F6	TERM処理3
D411	「ON(割り込み)GOSUB」文の行番号	D608～D60C	TERMのオペランド省略時のデータ
	より割り込みテーブルにテキスト番地を設定		
D41D	「GOSUB」チェック	D60D～D614	TERMの各処理へのジャンプテーブル
D426	COMnの割り込みテーブルの頭出し		
2A	同上(Bレジスタにポート番号)	D615	プリンタ クローズルーチン
33	同上(テキストから式の評価をして)	D617	プリンタ 1バイト出力ルーチン
D444	COMエントリ	D62B	改行用プリンタ1バイト出力ルーチン
D448	COMMON	D64A	プリンタ改行ルーチン
D459	COM ON/OFF/STOPエントリ	D659	プリンタのレディチェックと1バイト出力
5B	BASIC割り込みのON/OFF/STOP	D670	プリンタルーチンのBIOSへのRCB
	制御を割り込みテーブルに設定する	～D677	データ(＄D670～…LPCHK,
64	OFF処理		＄D672～…LPOUT)

アドレス	機　能　・　解　説	アドレス	機　能　・　解　説
D678	Abort ルーチン	DA36	SUBOUT　PUT後続設定ルーチン
D6AD ∫ D6B4	「Abort」メッセージ	DA62	BIOS実行ルーチン（$DA36付）
		64	BIOS実行ルーチン
D6B5	Abort処理実行ルーチン	DA6D	カーソルのX，Y座標設定ルーチン
D6D8	カセット1ブロック入力（汎用） ルーチン	DA80	「CLS」画面消去ルーチン
		8B	CLS実行ルーチン
D730	カセット1バイト入力（直接）ルーチン （エラーがなくなるまで）	DAA6	ブロック入力ルーチン
		B2	ブロック入力ルーチン （ユーザ用エントリ）
D735	カセット1バイト入力（直接）ルーチン （エラー時は強制リターン）		
		DADF	カーソル消去ルーチン1
D73E	カセット1バイト入力（直接）ルーチン	E4	カーソル表示ルーチン1
D758	MOTOR	DAEF	カーソル消去ルーチン2
D76F	モータON（モータ状態フラグが変る）	F6	カーソル表示ルーチン2 （コンソールコントロール）
72	モータOFF（モータ状態フラグが変る）		
78	モータON	DB07 ∫ DB0C	カーソル消去・表示ルーチン2の RCBデータ
7B	モータOFF		
D78C	カセット　ギャップ出力ルーチン	DB0D	キーボードジャンプテーブル
D794	カセット　1ブロック出力（汎用） ルーチン	DB1F	キーボードオープンルーチン（インプットモードでないとエラーにする）
D7DD	条件モータOFF	DB26	BEEP　ON/OFFのRCBデータ （$DB26…ON，$DB28…OFF）
D7E3	カセット1バイト出力（直接）ルーチン		
D7F7	一行入力（スクリーンエディタ） ルーチン	DB2A	BEEP
		DB49	一定時間BEEPを鳴らす （BELよりは短い）
D807	一行入力ルーチン（継続入力をしない）		
D8AC	EDIT	DB54	キーボード1バイト入力ルーチン
D90F	ブロック出力ルーチン	DB59	Breakチェックルーチン
D92F	フィールド設定ルーチン	DB6D	キー入力ルーチン
D93D	フィールド設定およびレーズルーチン	DB84	INKEY$
D944	行番号＋スペース出力	DB94	KEY ～ エントリ
D94A	AUTO編集処理	DB98	KEY　LIST
D980	スクリーンジャンプテーブル	DBD5	コンソール制御ルーチン　エントリ1
D992	スクリーンポジションチェックルーチン	D8	コンソール制御ルーチン　エントリ2
D99E	スクリーン　オープンルーチン （アウトプットモードでないとエラー）	DD	コンソール制御ルーチン　エントリ3
		E1	コンソール制御ルーチン　エントリ4
D9A5	プリンタオープンルーチン	DBF3 ∫ DBFA	PF・キー読み込み・表示ルーチン RCBデータ
D9D9	スクリーン1バイト出力ルーチン		
D9DE	カーソルのX，Y座標読み取りルーチン	DBFB	PF・キー読み込み・表示ルーチン
D9F4	汎用RCB設定ルーチン	DC15	KEY（n）ON/OFF/STOP エントリ
DA01 ∫ DA20	汎用RCB設定ルーチンのRCBデータ	35	KEY（n）ON処理
DA21	SUBOUT　PUT設定ルーチン	43	KEY（n）OFF処理
DA30	SUBOUT　PUT初期設定ルーチン	51	KEY（n）STOP処理

アドレス	機能・解説
DC56	PF・キー割り込み選択ルーチン（Dreg→IC）
DC6C	PF・キー割り込みON/OFF（未使用）
DC71	PF・キー定義ルーチン
DCA6	ONのエントリ
AA	ON KEY ～ ルーチン
BC	KEY（ ）の"（"＋数値＋"）"の読み取りルーチン
C5	KEYの割り込みテーブルの頭出しルーチン
CF	ON INTERVAL～
DB	ON TIME ～
EF	ON PEN ～
DCFA	CONNECT
DDA8	SYMBOL
DE33	GCURSOR
DECA	変数型と変数実効アドレスを GCURSORのワークエリアに代入
DEE2	Function機能チェックルーチン（NOTも含めて）
E4	Function機能チェックルーチン
DEF9	ワークエリア（Xレジスタの示す）にカーソル座標を代入
DF13	BIOS RCB, SUBOUT 設定ルーチン
DF27	BIOS RCB, SUBIN 設定ルーチン
DF30	X，Y座標の設定（式の評価，シザリング付き）
37	X，Y座標の設定（シザリング付き）
DF54	PAINT境界色設定ルーチン
DF78	POINT
DFA6	PSET
AC	PRESET
E014	X，Y座標（キャラクタ）のシザリング（X，YとX'，Y'について）
E019	X，Y座標（キャラクタ）のシザリング（X，Yについて）
E036	▼(X，Y)−(X，Y)▼ または ▼−(X，Y)▼ を読み込みX，Yを決定
E05B	▼(X，Y)▼ 読み込みX，Yを決定
E064	LINE
E11A	TIME（右辺にあって秒を与える）

アドレス	機能・解説
E15C	TIME$（右辺）
E172	年，月，日，時間，分，秒の個々の要素を設定
E17B	年，月，日，時間，分，秒の複数の要素を設定
E182	DATE$（右辺）
E190	DATE（右辺にあって日数を与える）
E1D8～E1E3	月の日数のデータ
E1E4	TIMEエントリ
E1F8	TIME$（左辺）
E20E	設定時刻が範囲外かどうかのチェック
E22B	時刻，分，秒などの10の位と1の位を足して計算しBレジスタへ
E233	TIME ON 処理
E23F	TIME OFF 処理
E249	TIME STOP 処理
E24F	TIME（割り込み設定）
E272	ワークエリアに割り込み時刻を読み込む
E283	SET TIMER（制御レジスタ）ルーチン
E291	DATE$（左辺）
E2CF	TIME$やDATE$の設定において範囲外のときの処理
E2DA	うるう年の確認ルーチン
E2E9	SET TIMER（24時間時計レジスタ）ルーチン
E2F9	E2E9の下位ルーチン（10の位と1の位を換算しサブコマンドに設定する）
E312	DATE$，TIME$のワークエリア設定ルーチン
E332	DATE$，TIME$，TIMEのパラメータ設定ルーチン
E34D	TIMERを読み込む〔(02F4～02F9)に格納〕
E36D	TIMERを(0102～)のワークエリアに読み込むルーチン
E377	TIMERを(01B5～01C9)のワークエリアに読み込むルーチン
E395	時刻などの数値変換
E3BA	INTERVALエントリ
E3C8	INTERVAL割り込み間隔設定ルーチン

アドレス	機能・解説	アドレス	機能・解説
E40B	INTERVAL ON処理	ED63	MMLバッファの頭出しルーチン
E43D	INTERVAL OFF処理	ED72	PLAY文割り込み処理ルーチン
E448	INTERVAL STOP処理	EE9A	音の長さ計算ルーチン
E44E	制御レジスタ（PSG）初期化ルーチン	EEC9	MML小文字⇨大文字変換
E453	0時割り込みルーチン	EEDA	"T"処理
E4BB	Color代入ルーチン	EEE5	"L"処理
E4C8	PAINT	EEF0	"O"処理
E520 〜 E543	CIRCLE用 SIN, COSデータ	EEFC	"V"処理
		FE	"S"処理
E544	CIRCLE X, Yの上限設定	EF12	オシレータディスクリプタへの書き込み
E552	CIRCLE エントリ	EF19	オシレータディスクリプタの読み出し
E6A3	CIRCLE 演算ルーチン	EF21	オシレータディスクリプタの頭出し
E6B1	CIRCLE 乗算ルーチン	EF31	"P", "R"処理
E7A3	8ビット→3ビット変換ルーチン	EF44	"A"〜"G"処理
E7EF	GET	EF5C	"N"処理
E80A	PUT	EFA1	"M"処理
EAF6	SCREEN関数	EFB3	変数名処理
EB4B	SCREEN	EFC9	音程の設定など
EB77	マルチ制御ページレジスタ（FD37）設定ルーチン	F01C	バッファにMML中間コードを格納
		3E	MML中間コード格納用サブルーチン
EB8E	BUBINI	F049	MML 1コード読み込み
EB91	BUBW	F060	ポインタ（D9：DA）の復帰
EB94	BUBR	F068	音程コマンドの後に続くものについての処理
EB9A	KILL		
EBBA	SOUND	F0CF	ポインタ（D9：DA）の設定
EBCF	PSG設定ルーチン	F0DA	音程以外のコマンドの後に続くものについての処理
EBE5	PSG初期化ルーチン		
EBF6	PLAY実行への設定	F0F5	数値，変数についての処理
EC05	音関係の初期化ルーチン	F144 〜 F159	コマンド相対アドレスデータ
EC14	TIMER割り込みイネーブル処理		
1B	TIMER割り込みマスク処理	F15A 〜 F174	音程のデータ
20	IRQマスク処理		
EC27 〜 EC28	BEEP OFFのRCBデータ	F176 〜 F17C	V1L1 '82 09 01（BIOSのバージョン）
EC29	PLAY文のオシレータの初期設定	F17D	BIOS
EC4D	PLAY文のバッファ初期設定	FC00	RAM領域
EC72	PLAY（PLAYのMMLは1つの言語体系を成している）	FC80	共有RAM領域
		FD00	I/O領域
ECD9	MMLバッファの初期設定	FE00	BOOT ROM
ECEE	MML文字領域チェック	FFE0	BOOT ROM用のRAM領域
ED19	MML解釈実行ルーチン	FFF0	割り込みベクトル
ED35	MMLコマンド解析ルーチン	FFFE	RESETベクトル

6－3　DISK BASICのメモリマップ

初期設定で使用される，IPLやDBSINIなどのルーチンも載せておきました．仕様は6－2と同様です．

アドレス	機能・解説	アドレス	機能・解説
0100～01A0	IPL（BASIC起動時に消滅）	74BD	ドライブテーブルの頭出しルーチン
		74CE	セクタ入出力ルーチン
0102～0121	IPLがBOOTROM内のサブルーチンを呼び出すためのデータ（RCB）	E6	セクタ入出力実行ルーチン
		7516	同上（RESTORE）
		2A	同上（DREAD）
6E00～71D5	DISK BASIC イニシャライズルーチン（BASIC起動後に消滅）	2D	同上（DWRITE）
		755D～7564	セクタ入出力実行ルーチンの各処理へのジャンプテーブル
6E03～6E43	DISKモードでのPF・キーのデータ	7565	DSKO$
7192～719B	DISK関係の予約語の予約語参照テーブルへの転送用データ	71	DSKI$，DSKO$入出力ルーチン
		7A	ドライブ，トラック，セクタ番号の設定
719C～71D5	DISK BASIC起動時のメッセージ	75A4	DSKI$
		75C9	KILL
		D4	KILL実行ルーチン
71D6～7313	DISK BASIC用のワークエリア	FB	FAT記入ルーチン
		762C	ディレクトリ探索ルーチン
7314～732F	DISKコマンドの予約語綴りテーブル	768E	ファイルの存在チェックルーチン
		7698	ファイルのオープンチェックルーチン
7330～733B	DISKコマンドジャンプテーブル	76BF	ID確認とFATの読み込みルーチン
		D2	同上（デバイス番号のチェックなし）
733C～735C	DISK関数の予約語綴りテーブル	7713	ディスク　オープンルーチン
		45	モード“I”のときの処理
735D～736E	DISK関数のジャンプテーブル	86	モード“O”のときの処理
		BE	モード“A”のときの処理
736F	DISK関係の解読実行ルーチン	7822	モード“R”のときの処理
73A1	DSKINI	783B	ファイルバッファの初期化ルーチン
7422	FAT初期化ルーチン	785F	ディレクトリスロット記入とFAT記入ルーチン
7445	コロン，NULLのチェックルーチン		
744C	「Are you sure～?」ルーチン	A9	FAT記入ルーチン（条件付）
745C～7468	「Are you sure」データ	78BE	ディスク　クローズルーチン
		7914	SFDからドライブ番号を割り出す
69	「Are you～?」実行ルーチン	7924	ディスク　1バイト出力ルーチン
749B～74A5	「(Y or N)?」データ	69	次セクタ設定ルーチン
		7990	空きクラスタ検索・設定ルーチン
74A6	メッセージ出力ルーチン	79AA	クラスタ番号，クラスタ内セクタ番号→トラック番号，セクタ番号変換ルーチン
74AB	ファイルバッファの頭出しルーチン		
B1	同上（任意のもの）	79C2	クラスタ番号→トラック番号－2

アドレス	機　能　・　解　説	アドレス	機　能　・　解　説
79D6	トラック番号－2→クラスタ番号	7C72	LOC
79DD	レコード／セクタ番号→クラスタ番号	7C7C	ファイルモードチェック（ランダム，テキストからファイル番号を読み込む）
79F5	クラスタ番号→レコード／セクタ番号		
79F9	ディスク　1バイト入力ルーチン	81	ファイルモードチェック（ランダム）
7A37	次セクタ読み込みルーチン	83	ファイルモードチェック（任意の）
86	ポインタをEOFまで進める	7CA4	ディスクEOF
7AA8	ポジションチェック	C3	ディスクLOF
7AB7	ディスクINPUTルーチン	7CE6	NAME
7B36	1バイト入力及びデータ終了チェック	7D31	"AS"チェック及びファイル名・ドライブ番号設定
3D	1バイト入力ルーチン		
7B43	区切文字設定ルーチン	33	ファイル名・ドライブ番号
7B57	FILES	7D43	"AS"チェック
7BEE～7BFD	「Clusters Free」データ	7D4E	FIELD
		7DA7	LSET
7BFE	1バイト＋空白出力	A8	RSET
7C01	空白出力	7E01	PUT
7C04	クラスタ数の割り出し	03	GET
7C15	DSKF	7EB0	DISK用エラー処理ルーチン
7C3D	CVI	7F25～7FC3	エラーメッセージ
40	CVS		
43	CVD	7FC4	DISKモード時の文字列代入用補助ルーチン
7C55	MKI$		
58	MKS$	7FE0～7FFF	未使用
5B	MKD$		

6－4　中間コード一覧表（コード順）

コードの小さい順に並べた，中間コードの対応表です．

（　　）でくくってあるのはサブコードです．また，［　　］の部分はDISKコードであり，DISKモード時のみ生成されます．

中間コード	対応するキーワード	エントリアドレス	中間コード	対応するキーワード	エントリアドレス
80	END	8FD0	(CE)	～ SUB	9039
81	FOR	A203	88	RUN	9021
82	NEXT	A2C7	(22)	～"(FD名)"	CF22
83	DATA	90A0	89	IF	90F1
84	DIM	94C5	8A	RESTORE	8FBC
85	READ	BFF3	8B	RETURN	9071
86	LET	9195	8C	REM	90A3
87	GO	902D	8D	'(注釈文字列)	90A3
(CD)	～ TO	9055	8E	STOP	8FD9

中間コード	対応するキーワード	エントリアドレス	中間コード	対応するキーワード	エントリアドレス
8F	ELSE	90A3	AD	EXEC	C76F
90	TRON	A01A	AE	OPEN	CEA8
91	TROFF	A01B	AF	CLOSE	CE73
92	SWAP	A784	B0	FILES	CBB5
93	DEFSTR	9F73	B1	COM	D444
94	DEFINT	76	(97)	～ ON	D475
95	DEFSNG	79	(D5)	～ OFF	D464
96	DEFDBL	7C	(8E)	～ STOP	D484
97	ON (01EC)→	DCA6	B1A2	COMMON D448→	90A0
(B2)	～ KEY GOSUB	DCAA	B2	KEY	DB94
(CB)	～ INTERVAL 〃	DCD3	(BB)	～ LIST	DB98
(FFA9)	～ TIME 〃	DCE5	(97)	～ ON	DC35
(FF9B)	～ PEN 〃	DCEF	(D5)	～ OFF	DC43
(B1)	～ COM (n) 〃	D40D	(8E)	～ STOP	DC4F
	～ (式) GO……	913A		～ 定義	DC71
(9B)	～ ERROR GOTO	A70F	B3	PAINT	E4C8
98	HARDC	ADDC	B4	BEEP	DB2A
99	RENUM	A9CA		～ 1	DB3F
9A	EDIT	D8AC		～ 0	DB44
9B	ERROR	A739	B5	COLOR	C7CA
9C	RESUME	A744	(E6)	COLOR＝	C7F1
9D	AUTO	9FE7	B6	LINE	E064
9E	DELETE	9FBC	(FFA6)	～ INPUT	BF48
9F	TERM	D52D	B7	DEF	A7D9
A0	WIDTH	C87B	(CF)	～ FN	A7DF
A1	UNLIST	C912	(D2)	～ USR	A969
A2	MON	ABF4	B8	POKE	9A30
A3	LOCATE	C8F0	B9	PRINT	9AB3
A4	CLS	DA80	(40)	～ @	9C2B
A5	CONSOLE	C814	BA	CONT	9002
A6	PSET	DFA6	BB	LIST	CD01
A7	PRESET	DFAF	BC	CLEAR	C77B
A8	MOTOR	D758	BD	RANDOMIZE	BBE5
(97)	～ ON	D76F	BE	WHILE	AD54
(D5)	～ OFF	D772	BF	WEND	AD81
A9	SKIPF	CAE6	C0	NEW	8F32
AA	SAVE	CD3E	C1	GET	E7EF
(4D)	～ M	CE04	(23)	～ #	7E04
AB	LOAD	CF31	C2	PUT	E80A
(4D)	～ M	D030	(23)	～ #	7E01
(3F)	～ ?	CAE4	C3	CIRCLE	E552
AC	MERGE	CF29	C4	CONNECT	DCFA

中間コード	対応するキーワード	エントリアドレス	中間コード	対応するキーワード	エントリアドレス
C5	SYMBOL	DDA8	E8	DSKINI	73A1
C6	GCURSOR	DE33	E9	DSKO$	7565
C7	BUBINT	EB8E	EA	NAME	7CE6
C8	BUBW	EB91	EB	FIELD	7D4E
C9	BUBR	EB94	EC	LSET	7DA8
CA	KILL	EB9A	ED	RSET	7DA7
CB	INTERVAL	E3BA	EE	CHAIN	808E
(97)	～ ON	E40B	EF	ERASE	8448
(D5)	～ OFF	E43D	F0	LLIST	CD88
(8E)	～ STOP	E448	F1	LPRINT	9A7F
CC	TAB ( ) 関数	9A50	F2	SOUND	EBBA
CD	TO	——	F3	PLAY	EC72
CE	SUB	——			
CF	FN関数	A82A	26	&定数	9E7E
D0	SPC ( ) 関数	9BB5	(4F)	&O (8進定数)	9E94
D1	USING	A07F	(48)	&H (16進定数)	9EA5
D2	USR関数	A9A0	FE01	1バイト整定数	(9212)
D3	ERL	9270	FE02	2バイト 〃	(921B)
D4	ERR	9264	FE04	単精度型定数	(9225)
D5	OFF	——	FE08	倍精度型定数	(〃)
D6	THEN	——	FEF2	行番号定数	(9205)
D7	NOT (否定演算子)	924E			
D8	STEP	——	FF80	SGN	B3BE
D9	+ (数値)	BD4D	FF81	INT	B472
	+ (文字列)	98AE	FF82	ABS	B3D5
DA	− (2項演算子)	BD3F	FF83	FRE	972C
	− (単項演算子)	92A7	FF84	POS	C92A
DB	*	BD73	FF85	SQR	BAEE
DC	/	93CD	FF86	LOG	B0AA
	/ (実行ルーチン)	B236	FF87	EXP	BB5B
DD	^	939E	FF88	COS	BE60
	^ (実行ルーチン)	BAF7	FF89	SIN	BE66
DE	AND	94A8	FF8A	TAN	BEB1
DF	OR	94AD	FF8B	ATN	BEF5
E0	XOR	94B2	FF8C	PEEK	9A26
E1	EQV	94B7	FF8D	LEN	992A
E2	IMP	94BC	FF8E	STR$	9752
E3	MOD (剰余)	BE33	FF8F	VAL	99CA
E4	Y (整数除算)	BE0C	FF90	ASC	9947
E5	> 比較演算子 (基地)	9433	FF91	CHR$	9933
E6	= 比較演算子 (数値比較)	BE44	FF92	CINT	BC74
E7	< 比較演算子 (文字列比較)	946B	FF93	CSNG	BCE0

中間コード	対応するキーワード	エントリアドレス
FF94	CDBL	BCCD
FF95	FIX	B461
FF96	SPACE$	9D7B
FF97	HEX$	9E2C
FF98	OCT$	9E2D
FF99	LOF	D11D
FF9A	EOF	D11A
FF9B	PEN (文)	0257
	〃 (関数)	025A
FF9C	LEFT$	9952
FF9D	RIGHT$	996F
FF9E	MID$ (文)	9DC1
	〃 (関数)	9979
FF9F	INSTR	9CD4
FFA0	SCREEN (文)	EB4B
	〃 (関数)	EAF6
FFA1	ANPORT	AED2
FFA2	VARPTR	A952
FFA3	STRING$	9D59
FFA4	RND	BBFE
FFA5	INKEY$	DB84
FFA6	INPUT (文)	BFDD
(24)	INPUT$関数	D15C
FFA7	CSRLIN	C923
FFA8	POINT	DF78
FFA9	TIME	E1E4
	～ "hh/ mm/ss"	E24F
(97)	～ ON	E233
(D5)	～ OFF	E23F
(8E)	～ STOP	E249
(24)	～ $ = "時分秒"	E1F8
FFA9	TIME関数	E11A
(24)	～ $ 〃	E15C
FFAA	DATE$ = "年月日" 文	E291
	DATE関数	E190
(24)	～ $ 〃	E182
FFAB	DSKF	7C15
FFAC	CVI	7C3D
FFAD	CVS	7C40
FFAE	CVD	7C43
FFAF	MKI$	7C55
FFB0	MKS$	7C58
FFB1	MKD$	7C5B
FFB2	LOC	7C72
FFB3	DSKI$	75A4
FFB4	LPOS	C947

6－5　実装図・回路図

本節では，ＦＭ－７のミニフロッピディスクインタフェース（ＭＦＤ）カード，Ｚ８０カード，ＲＳ－２３２Ｃインタフェースカードの回路図・実装図を紹介します．

これらの回路図・実装図はすべて秀和システムトレーディング株式会社において調査したもので，メーカーが保障したものではありません．あくまでも参考図としてお使い下さい．従って，本回路図集に関してメーカー等に直接問い合わせることは御遠慮下さい．

I　実装図

1　ミニフロッピディスクインタフェース（ＭＦＤ）カード
2　Ｚ８０カード
3　ＲＳ－２３２Ｃインタフェースカード

II　回路図

1　ミニフロッピディスクインタフェース（ＭＦＤ）カード
2　Ｚ８０カード
3　ＲＳ－２３２Ｃインタフェースカード

※　印のない抵抗の単位はすべてΩです．
ＫはkΩ，μはμＦ，ＰはpＦの略です．
＊印は負論理のものです．

I・1　MFD　カード　実装図

コネクタCN2：
FCN-704P034-AU/M♯00A

コネジ：2-F6-SW4NA-2.6×10S

カナグ：2-FCN-360A3

コネクタCN1：FCN-365P032-AG

I・2　Z80　カード　実装図

カナグ：FCN360A3
コネジ：F6-SWANA-2.6×10S

コネクタCN1：FCN365P040-AG

I・3　RS232C　カード　実装図

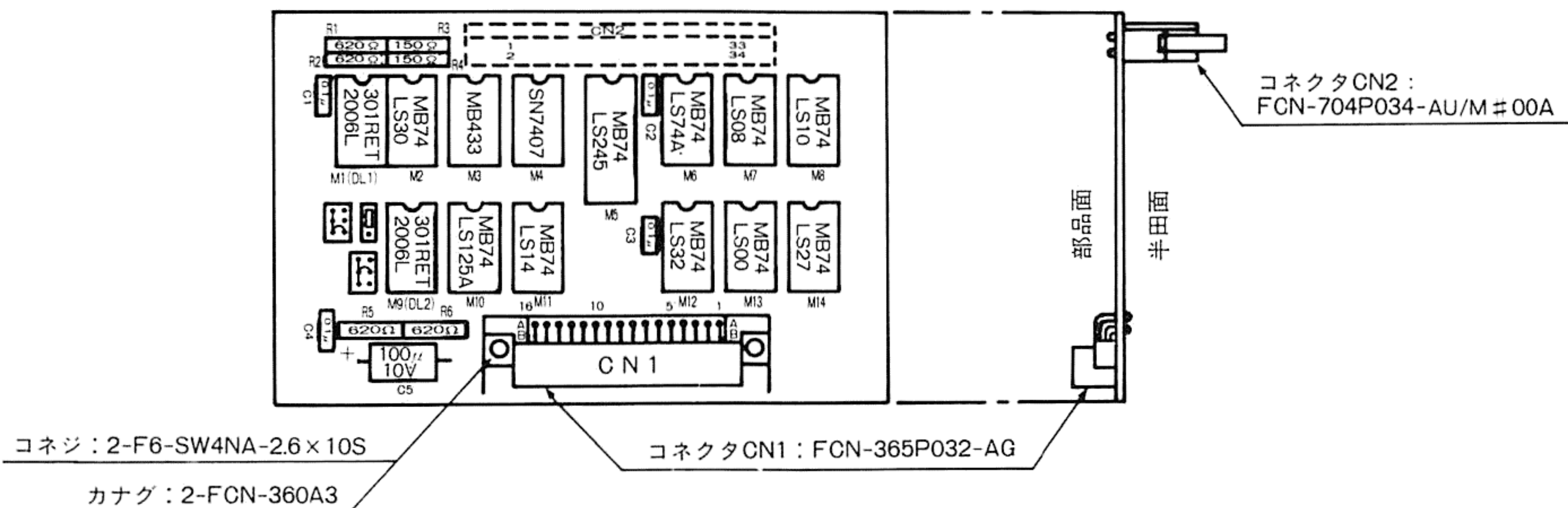
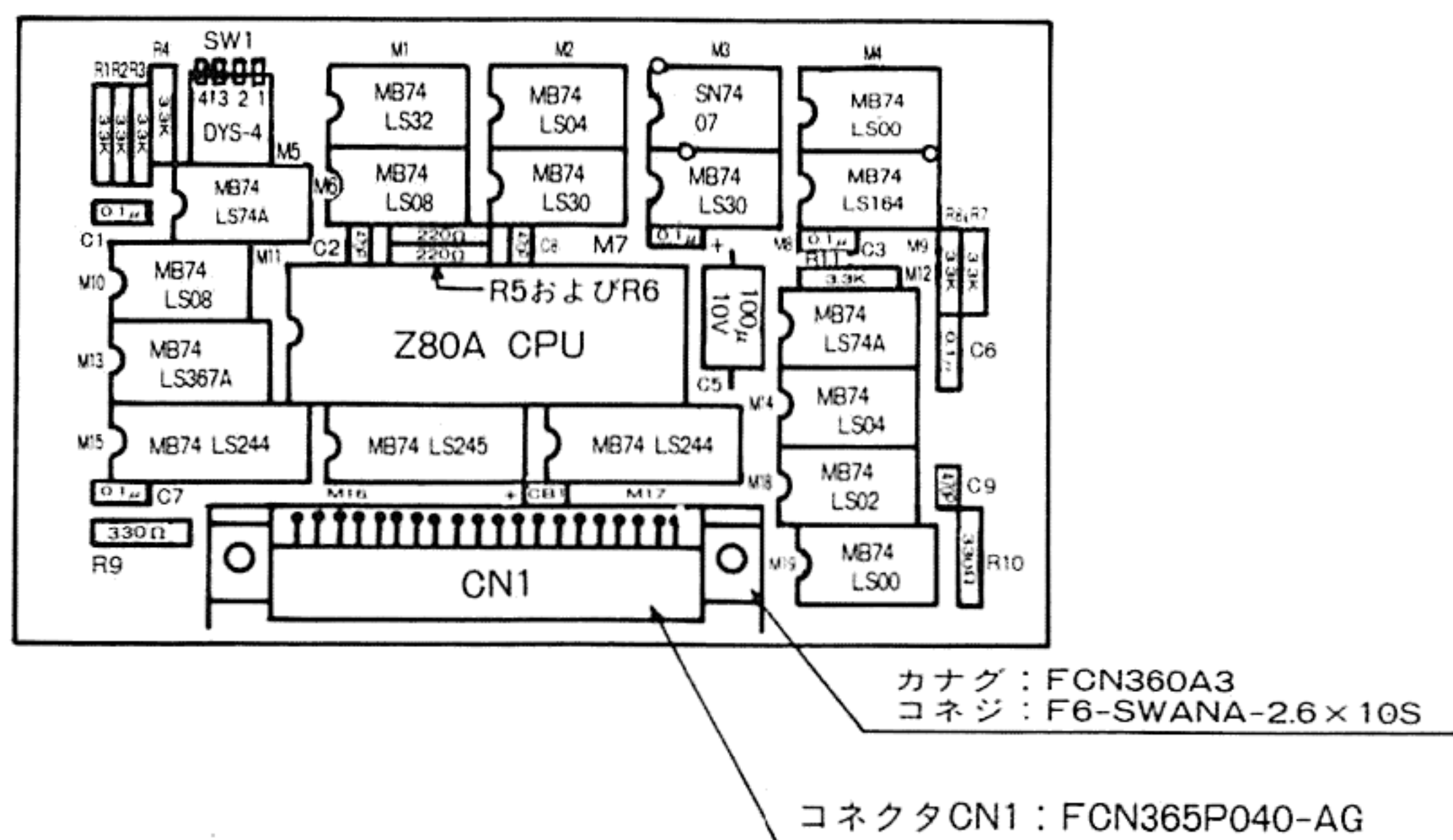
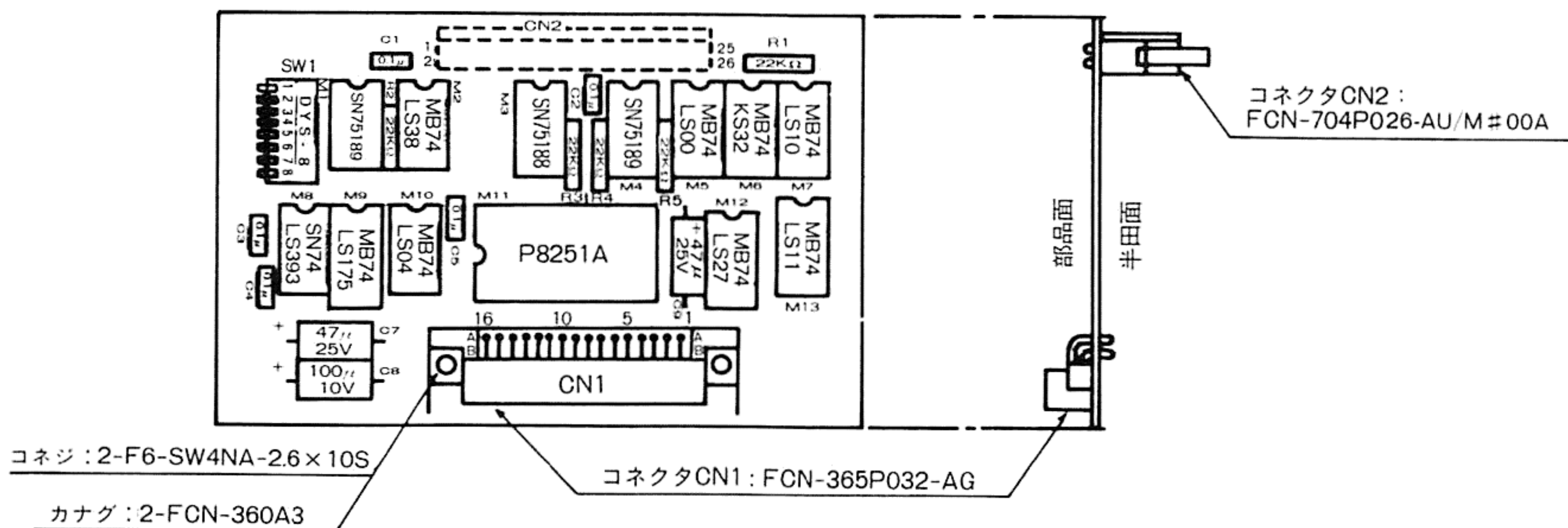
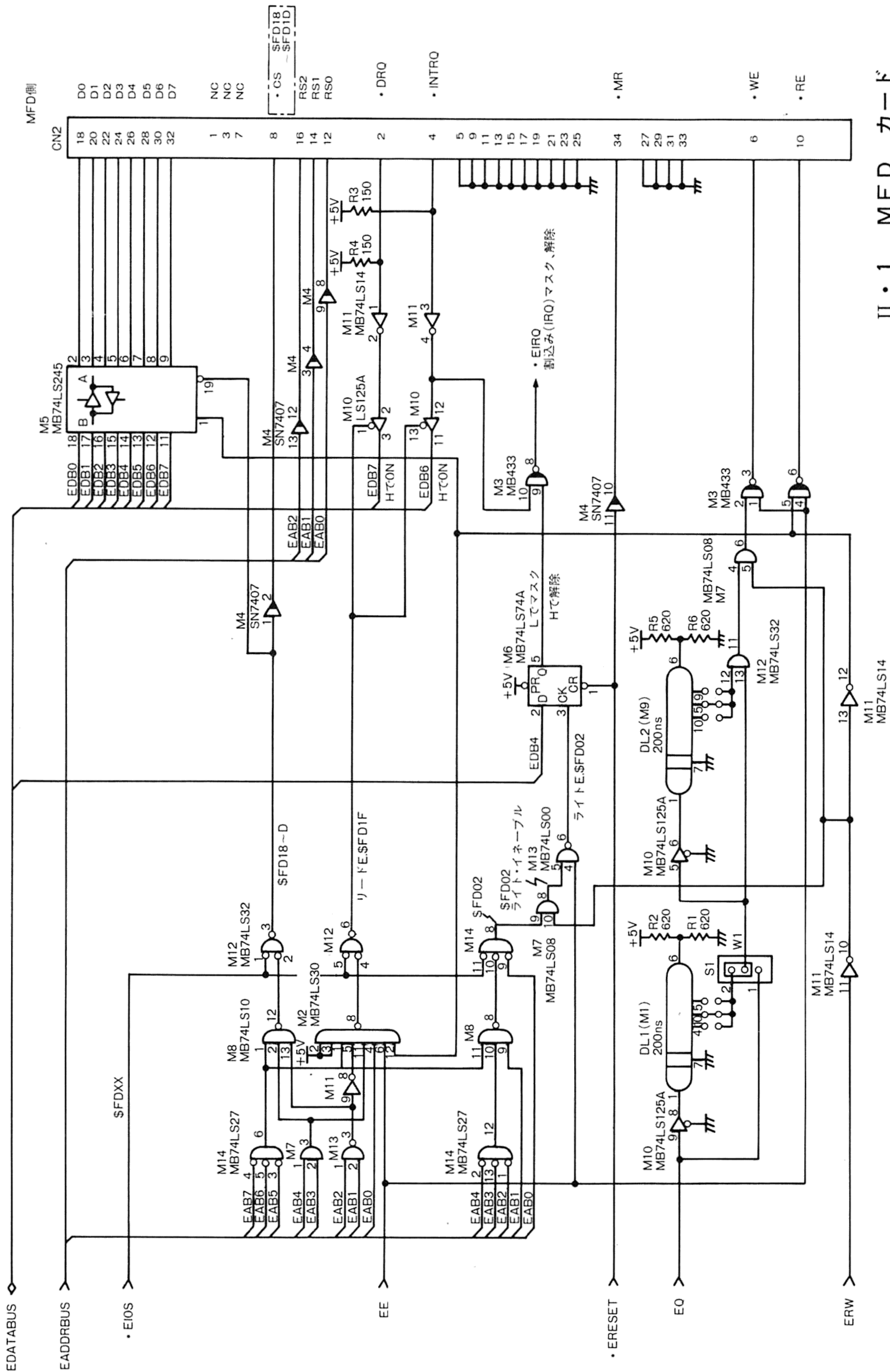
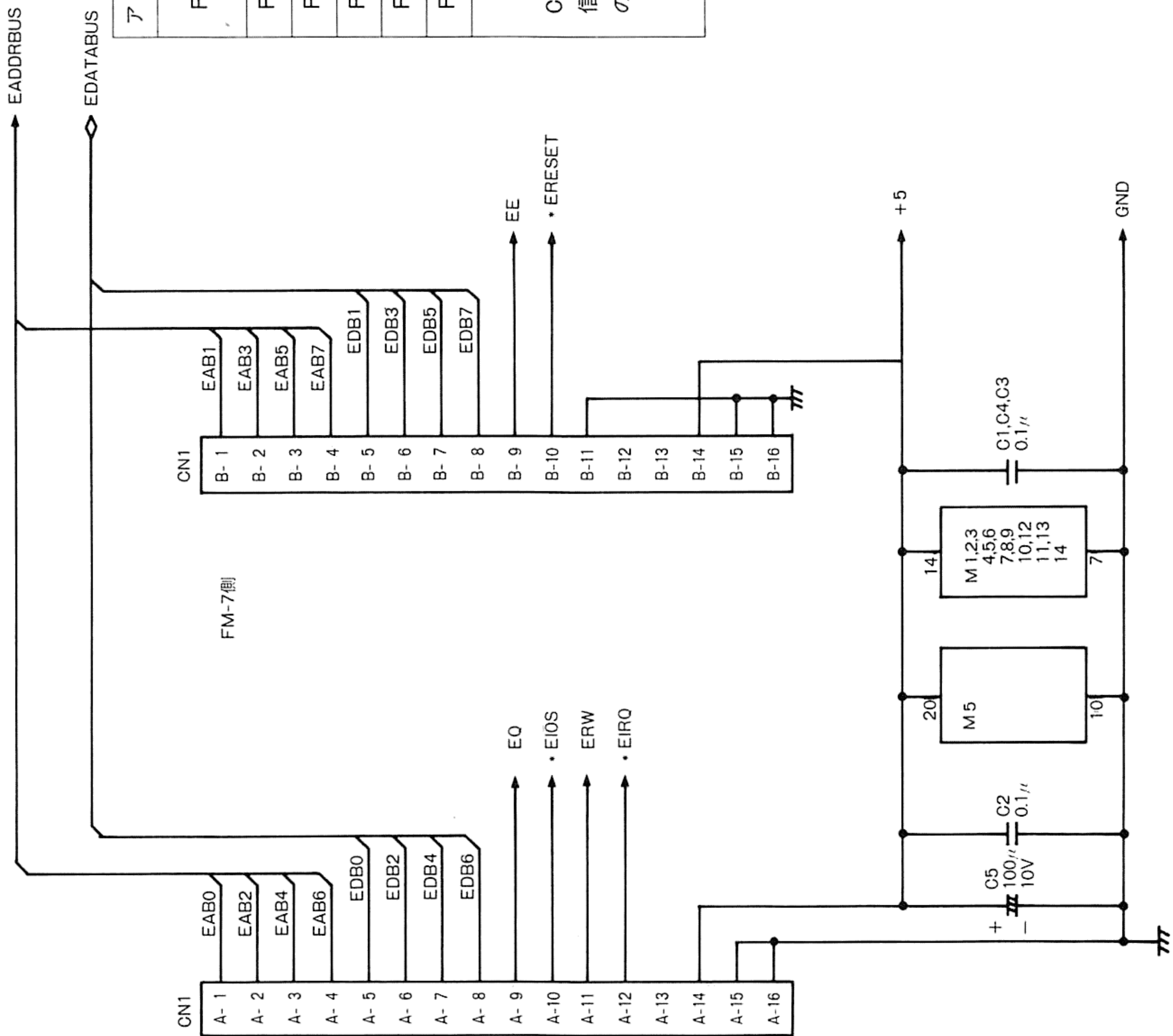
コネクタCN2：
FCN-704P026-AU/M♯00A

コネジ：2-F6-SW4NA-2.6×10S

カナグ：2-FCN-360A3

コネクタCN1：FCN-365P032-AG

II・1　MFD　カード

II・1 MFD カード

アドレス	リード／ライト	レジスタ	説明
FD18	リード	ステータス	コマンド終了時のエラー状態を示す
	ライト	コマンド	コマンドのセットをする
FD19	リード/ライト	トラック	R/Wヘッドの現在トラック位置を保存しているレジスタ
FD1A	リード/ライト	セクタ	目的セクタアドレスをロードする
FD1B	リード/ライト	データ	リード・ライト時にデータ保持レジスタとして使用される
FD1C	リード/ライト	ヘッド	目的ヘッド(ディスクのSide)が0か1かを選択する
FD1D	リード/ライト	ドライブ	目的ドライブナンバを選択する
CN2の信号線の意味	CPU←$\overline{DRQ}$―FDC		ディスクのリードライト時、1バイトごとに"L"になりR/W信号によりセットされる
	CPU←$\overline{INTRQ}$―FDC		コマンドが終了されると"L"になる
	CPU―$\overline{MR}$→FDC		YD2740内の各デバイスをリセットし、コマンドレジスタのクリアを行う

CN1 A-1 EAB0, A-2 EAB2, A-3 EAB4, A-4 EAB6, A-5 EDB0, A-6 EDB2, A-7 EDB4, A-8 EDB6, A-9 EQ, A-10 ・EIOS, A-11 ERW, A-12 ・EIRQ, A-13, A-14, A-15, A-16

FM-7側

CN1 B-1 EAB1, B-2 EAB3, B-3 EAB5, B-4 EAB7, B-5 EDB1, B-6 EDB3, B-7 EDB5, B-8 EDB7, B-9 EE, B-10 ・ERESET, B-11, B-12, B-13, B-14, B-15, B-16

EADDRBUS

EDATABUS

+5

GND

C5 100μ 10V

C2 0.1μ

M5 (20, 10)

M1,2,3, 4,5,6, 7,8,9, 10,12, 11,13, 14 (14, 7)

C1,C4,C3 0.1μ

部品面 CN1 (A, B, 16, 1)

CN2 半田面 (31, 34, 1, 2)

II・2 Z80 カード

CN-1
A- 1 AB 0
A- 2 AB 1
A- 3 AB 2
A- 4 AB 3
A- 5 AB 4
A- 6 AB 5
A- 7 AB 6
A- 8 AB 7
A- 9 AB 8
A-10 AB 9
A-11 AB10
A-12 AB11
A-13 AB12
A-14 AB13
A-15 AB14
A-16 AB15
A-17 Z80W($FD05 DB0)
A-18 BSB
A-19 BAB
A-20 RWB
MADDRBUS

CN-1
B- 1 DB0
B- 2 DB1
B- 3 DB2
B- 4 DB3
B- 5 DB4
B- 6 DB5
B- 7 DB6
B- 8 DB7
MDATABUS
B- 9 Z80φ
B-10 • RESETB
B-11 • IRQ
B-12 • NMI
B-13
B-14
B-15 • FIRQ
B-16 EB
+5V
B-17
B-18 • REFCK
B-19
B-20 QB

+5V
20 M15,16,17 10
16 M13 8
14 M2,3,4,5 6,7,8,9 10,11, 12,14 7
C5 100μ 10V
C8 0.1μ
C1,4,6,7 0.1μ
GND

II・2 Z80 カード

II・3　RS232C　カード

II・3　RS232C カード

共　著　　箕原辰夫　　菊地　寿　　南　泰浩

発行者　牧谷秀昭
発行所　秀和システムトレーディング株式会社
　　　　郵便番号１０７　東京都港区南青山２－１９－５　関原ビル

SHUWA SYSTEM TRADING CO., LTD.
SEKIHARA BLDG,
2-19-5 MINAMIAOYAMA, MINATO-KU, TOKYO, 107 JAPAN

印刷所　東京スガキ印刷株式会社

1983年12月16日第１刷発行　1985年２月１日第２刷発行